USAISEC
US Army Information Systems Engineering Command
Fort Huachuca, AZ 85613-5300

(1)

U.S. ARMY INSTITUTE FOR RESEARCH
IN MANAGEMENT INFORMATION,
COMMUNICATIONS, AND COMPUTER SCIENCES

AIRMICS

Integrated Office Information System (IOIS) Summary Report:

Automated Session Manager Analysis, Design, and Implementation

ASQB-GM-90-023

MAY 1990

AIRMICS
115 O'Keefe Building
Georgia Institute of Technology
Atlanta, GA 30332-0800

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188
Exp. Date: Jun 30, 1986

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT N/A
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ASQB-GM-90-023	5. MONITORING ORGANIZATION REPORT NUMBER(S) N/A
6a. NAME OF PERFORMING ORGANIZATION AIRMICS 6b. OFFICE SYMBOL (If applicable) ASQB-GM	7a. NAME OF MONITORING ORGANIZATION N/A
6c. ADDRESS (City, State, and Zip Code) 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, Ga 30332-0800	7b. ADDRESS (City, State, and ZIP Code) N/A
8b. NAME OF FUNDING/SPONSORING ORGANIZATION AIRMICS 8b. OFFICE SYMBOL (If applicable) ASQB-GM	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 115 O'Keefe Bldg. Georgia Institute of Technology Atlanta, GA 30332-0800	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 62783A; PROJECT NO. DY10; TASK NO. 05; WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Integrated Office Information System (IOIS) Summary Report:
Automated Session Manager Analysis, Design, and Implementation

12. PERSONAL AUTHOR(S)
Dr. Jay F. Nunamaker, Jr., Dr. Olivia R. Liu Sheng, Milam W. Aiken

13a. TYPE OF REPORT	13b. TIME COVERED FROM_____ TO_____	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
		May 1990	115

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUBGROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

The object of this research has been to study and build a prototype of an Integrated Office Information System (IOIS). The need for this system was derived from problems in existing automated office systems including: the lack of standard software tool user interfaces, excessive use of paper to transmit information, redundant, and inconsistent information in common databases, and haphazard coordination and control of information exchange. The IOIS which was developed addressed these problems by providing the means to accomplish a variety of tasks while maintaining data integrity, reducing the flow of paper, improving communication, and standardizing user interfaces and software protocols.

The Automated Sessions Manager (ASM), a component of IOIS and the focus of this report, was developed to facilitate meetings of organization group members working in distributed offices. ASM facilitates the use of a suite of group support tools known as the Asynchronous Group Decision Support System (AGDSS) also discussed here. The architecture of ASM consists of expert systems for pre-session planning, session facilitation, and post-session analysis as well as group status, calendar scheduling, and data integration facilities. This report presents a detailed summary of research efforts involved in the design and implementation of these features.

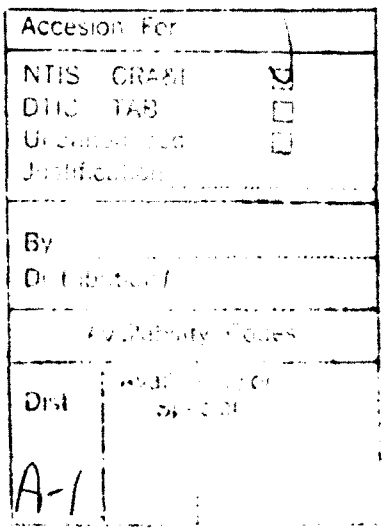
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
[X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL
Michael Evans	404/894-3107	ASQB-GM

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

This research was performed for the Army Institute for Research in Management Information, Communications and Computer Science (AIRMICS), the RDTE organization of the U.S. Army Information Systems Engineering Command (USAISEC). This research is not to be construed as an official Army position, unless so designated by other authorized documents. Material included herein is approved for public release, distribution unlimited. Not protected by copyright laws.



THIS REPORT HAS BEEN REVIEWED AND IS APPROVED

s/ James Gantt
James Gantt
Chief, MISD

s/ John R. Mitchell
John R. Mitchell
Director
AIRMICS

Integrated Office Information System (IOIS) Summary Report: Automated Session Manager Analysis, Design, and Implementation †

Dr. Jay F. Nunamaker, Jr.
Dr. Olivia R. Liu Sheng
Milam W. Aiken
Department of Management Information Systems
College of Business and Public Administration
University of Arizona
Tucson, Arizona 85721
602-621-2748

May 2, 1990

[0]†Submitted to the Army Institute of Research in Management Information, Communications and Computer Science (AIRMICS), Atlanta, GA. Grant #: DAKF-11-88-C-0021.

Abstract

The objective of this research project has been to study and build a prototype of an Integrated Office Information System (IOIS). The need for this system was derived from problems in existing automated office systems including: the lack of standard software tool user interfaces, excessive use of paper to transmit information, redundant and inconsistent information in common databases, and haphazard coordination and control of information exchange. The IOIS which was developed addressed these problems by providing the means to accomplish a variety of office tasks while maintaining data integrity, reducing the flow of paper, improving communication, and standardizing user interfaces and software protocols.

The Automated Session Manager (ASM), a component of IOIS and the focus of this report, was developed to facilitate meetings of organization group members working in distributed offices. ASM facilitates the use of a suite of group support tools known as the Asynchronous Group Decision Support System (AGDSS) also discussed here. The architecture of ASM consists of expert systems for pre-session planning, session facilitation, and post-session analysis as well group status, calendar scheduling, and data integration facilities. This report presents a detailed summary of research efforts involved in the design and implementation of these features.

Table of Contents

Contents

1 Introduction

The Integrated Office Information System (IOIS) is an advanced office architecture developed for and funded by the Army Institute of Research in Management Information Communications and Computer Science (AIRMICS). The development of this architecture was motivated primarily by the absence of adequate integrated office frameworks for the office. Existing frameworks consist of procedural and form automation systems for non-managerial office workers and generic tools (such as spreadsheets) for managerial and professional personnel. Concepts to support different levels of workers, such as tailoring generic tools for office environments, using common interfaces, and providing AI techniques have not been addressed within a single framework prior to the development of IOIS. Reasons for developing an integrated architecture rather than using isolated systems for clerical and managerial workers include: (1) an integrated architecture will reduce maintenance and allow better control of office and system operations, (2) sharing of common data and knowledge bases will contribute to reduced redundancy and improve application development time, and (3) common interfaces will reduce training costs.

The components for the IOIS architecture were selected based on literature stating the requirements of managerial, professional and clerical personnel [2, 3]. For managers, providing support for decision making, meetings, and resource management was considered important, while professional and clerical workers primarily required access to databases, form management systems, and conventional office tools. The initial IOIS framework included a resource management expert, an asynchronous group decision support system (AGDSS) facility, some decision support systems, a calendar scheduler, and some conventional office tools with a common knowledge base/database and interface providing the integration.

The focus of this report is to detail the development of the Automated Session Manager (ASM) subsystem which supports the AGDSS facility. While the calendar scheduler module is used by several other subsytems, it is also discussed in this report since it is an integral component of the process for planning AGDSS meetings.

2 IOIS Project Synopsis

The analysis, design, and implementation of the automated session manager must be viewed in the larger context of the overall project. A short synopsis of the IOIS project history is presented which

Mar 1988	Pilot system for IOIS.
Aug 1988	IOIS architecture completed.
Sep 1988	First version of AIRMICS case study, hardware requirements.
Sep 1988	ASM analysis phase.
Oct 1988	Received Nexpert Object 1.0.
Dec 1988	ASM design phase.
Jan 1989	Received Nexpert Object 1.1.
Feb 1989	ESP tool selection implemented.
Feb 1989	ESP group selection implemented.
Mar 1989	M. Morrison attended Bechtel Nexpert seminar.
Apr 1989	M. Morrison attended Nexpert user group meeting.
Apr 1989	Text-based interface designed.
Apr 1989	Distributed EBS and IA implemented.
Jun 1989	Final IOIS prototype completed.
Sep 1989	IOIS prototype delivered to AIRMICS.

Table 1: IOIS Project Schedule in Retrospect

indicates how research on ASM fits into this overall context.

A study of the AIRMICS office was conducted to provide a target implementation environment for the overall system. A prototype of IOIS was subsequently delivered to AIRMICS in September 1989 and included expert systems for GDSS tool and group selection, a meeting scheduler, resource management software, and user-friendly computer interface features. A synopsis of the project history appears in *(Table 1)*.

3 Automated Session Manager Analysis

Since AIRMICS conducts and sponsors research, it develops short-and long- term research plans. The function of identifying research areas and conducting research is referred to as RDTE (Research,

Development and Technical Evaluation). In the process of producing a procurement package (identified in an earlier report [4]), AIRMICS may use group support tools when personnel wish to identify areas of interest for possible funding and when a decision is made on the projects to fund. Also, when research projects terminate, a final IPR meeting is conducted, again requiring meeting support tools.

The Asynchronous Group Decision Support System (AGDSS) tools (discussed in *Appendix I* and *Appendix II*) provide an ideal environment in which to conduct these meetings since participants may be distributed geographically (in different offices) and temporally (all participants need not be 'logged-on' at the same time). The AGDSS tools could effectively augment the process of identifying the areas of interest, selecting the projects to be funded, and developing a research plan. By providing automated record keeping of meeting notes, allowing anonymous input of ideas, and supporting the office worker at his convenience (in his office when he has spare time), AGDSS should greatly increase the productivity of these meetings. *Appendix III* provides additional information regarding distributed group meetings and special design requirements.

Since selecting and using the AGDSS tools, selecting group participants, scheduling the times to meet, and facilitating the resulting meeting often requires considerable expertise, however, AIRMICS could benefit from an Automated Session Manager which alleviates much of the burden normally placed upon a human group facilitator. Consequently, group members with only limited experience in facilitation could, with the assistance of ASM, assume the role of group facilitator.

During the ASM analysis phase, a panel of local GDSS experts met over a period of three months in the fall of 1988 to determine the factors involved in the tool and group selection process. This panel of experts included group facilitators with several years of GDSS session experience and tool developers and researchers. In the final stages of analysis, a GDSS session involving issue analysis and organization was conducted at the University of Arizona to categorize the selection factors involved. The team members also met with Major Ted Hengst from AIRMICS to determine the tools and participants needed for meetings based on job responsibilities, work interests, and departmental affiliations. This knowledge was incorporated into the system and is discussed in more detail in the design section below.

4 Automated Session Manager Design

The analysis phase of ASM indicated the need for several different modules which would be called in sequence: a pre-session planning facility (ESP), an expert session facilitator (ESF) to help the facilitator manage the meeting, and an expert session analyzer (ESA) to help the facilitator make sense of meeting results.

4.1 Expert Session Planner

Expert Session Planner (ESP) is designed specifically to assist electronic meeting coordinators in pre-session planning of group meetings. ESP can assist the meeting coordinator in the selection of appropriate tools through the ESP(tool) expert system and the selection of participants for the session through the ESP(group) expert system. In addition, ESP can scan the calendars of the session participants to determine convenient times for having face-to-face meetings through the use of the ESP(sched) subsystem. ESP thus allows planning and coordinating of electronic meetings with minimal knowledge of the GDSS tools and the processes involved in pre-session facilitation.

Once the underlying knowledge of the expert task has been acquired in the analysis phase, a theoretical representation of the knowledge must be developed. Some representations include: IF-THEN rules (perhaps the simplest and most prevalent technique), Frames and Objects (popular for their powerful inheritance and hierarchical properties), and Semantic Networks (noted for their flexible, yet powerful structures). The choice of knowledge representation is perhaps the guiding force of the ultimate implementation.

An expert system for pre-session planning will benefit from a Frame and Object representation since group and tool characteristics often have hierarchical structures. For example, group participants can be classified by organizational affiliation, interests, or responsibilities. An expert system which selects appropriate group participants can rely on certain inherited features instead of features which are expressed for each group member. Likewise, GDSS tools can be classified by their applicabilibilty for a particular group or problem. Inheritance of these characteristics will allow the expert system to proceed more efficiently in determining which GDSS tools to select.

The initial prototype of ESP was designed with the procedures of selecting the tool and participants represented in IF-THEN rules with the facts about the tools and participants stored in a database. When the rules are fired, the facts associated with the objects in the rules are instantiated from the database.

This reduces the number of rules required in both selecting the tools and participants. The knowledge base can be further refined with the use of powerful representation techniques such as semantic nets, frames, and object-oriented approaches, to be investigated in future.

The expert task must also be analyzed as to the fundamental nature of the problem: one of design, diagnosis, or both. Design problems typically feature a relatively limited set of underlying conditions and a much greater number of goals; diagnosis problems include relatively few goals with many conditions which must be verified. For example, the problem of group selection is one of design since the number of potential participants is likely to be much greater than the number of criteria involved in the group configuration. Conversely, tool selection is primarily a matter of diagnosis; there are a relatively limited number of tools while there are many more factors involved in their selection. A forward-chaining inferencing strategy is most efficient with design problems, while a backward-chaining strategy is most economical with diagnosis tasks. Some problems may involve diagnosis and design, indicating a need for a hybrid inferencing strategy.

ESP(tool) uses an exhaustive, backward-chaining inferencing strategy with provisions for rule uncertainty to resolve conflicting rules. ESP(group) uses an exhaustive, forward-chaining inferencing strategy which configures a query which is then used by a dBase III + database management system to search a database of user profiles, looking for appropriate meeting participants.

The calendar scheduler was designed for use through the IOIS project. It can be used in a batch mode or in a stand-alone mode. In the batch mode, it is used by the Session Planner to schedule GDSS sessions. In the stand-alone mode it can be used to schedule meetings, resources, or appointments. Although a number of calendar tools are commercially available, none provided the capability to incorporate intelligence or supplied a programming interface that could be integrated with the IOIS. Therefore, the calendar scheduler was designed and developed with the following capabilities:

1. Monitor project activities,
2. Schedule meetings,
3. Specify user profiles (e.g. times convenient for meetings),
4. Schedule resources, and
5. Schedule appointments.

4.2 Expert Session Facilitator and Expert Session Analyzer

Details of the design for each of these expert systems were not pursued due to time limitations. However, an early form of ESF has been tested with the AGDSS tools which kept a count of user comments with some rudimentary statistics on frequency, averages, etc. Another non-expert form of ESF (discussed in part in *Appendix III*) shows how facilitation can be supported through a group status indicator.

Very little has been done to automate the session analysis phase of group meetings, however.

5 Automated Session Manager Implementation

5.1 Expert Session Planner Implementation

Only the ESP component of ASM was implemented due to time constraints. The use of ESP follows the sequence below:

1. **Group Selection Module**

   The meeting coordinator begins ESP with the group selection module to determine the type of the meeting, the topic of the meeting, and other information. This information is first inferred from the knowledge base which fires appropriate rules and determines more information from the meeting coordinator as it 'back-chains' through the knowledge base. Once it has determined the meeting type and topic, the group selection module instantiates the user profile knowledge from the data base and selects participants who should participate in the meetings based on the facts from their profile. Personnel interests, responsibilities, and organizational affiliations are all factors determining membership in a potential group meeting.

   The meeting topic helps in determining the initial list of participants. For instance, if the topic involves discussion on a project then all members responsible for the project are selected. Further, the supervisor of the project will be selected as well as any member who has listed interest in the area of the project. The meeting type, on the other hand, helps in determining who should be added or deleted from the list. For instance, if it is a peliminary discussion meeting, then the supervisor will be removed, or if the discussion involves financial matters, then the financial officer involved with the project will be added. This process is iterated until the knowledge base

is exhausted and a final list is presented to the meeting coordinator. The meeting coordinator can use the list as is or make further changes depending on his preferences. The role of group selection is merely to provide a starting point for the meeting coordinator.

The group determination module is implemented using EXSYS Professional, an expert system shell from EXSYS, Inc. The knowledge base consists of 14 rules which composes a query for a dBase III + database management system (DBMS). This DBMS then selects organizational personnel for the meeting based upon user profiles in a dBase III + database and configures a participant list. This list is then written to an ASCII file for later reference.

2. **Calendar Scheduling Module**

The calendar scheduling module is strictly used for meeting sessions involving face-to-face interaction. Once the participant list is determined, this module checks the calendar information of each of the participants selected as well as the calendar of the meeting room to arrive at a final time for the session. Currently, the calendar program is simplistic in the sense that it selects the first available time for the participants. Conflict resolution techniques will be included in future implementations of this module.

This module is currently implemented in Turbo Pascal 4.0 and interacts with a calendar program to access the personal schedules of the participants.

3. **Tool Selection Module**

Once a group membership list has been constructed and meeting times have been arranged, the meeting coordinator used the tool selection module to determine the GDSS tools required for the session. The coordinator must have already used the group selection module at this point because he is expected to answer questions regarding this group's characteristics. For example, the coordinator must know the extent of the participants' familiarity with the topic as well as the extent of common knowledge among the group members. Additonal questions are asked concerning problem characteristics such as whether or not the problem can be segmented and whether or not stakeholder identification is important. Once all of the questions are answered, a list of recommended tools is written to a file with their accompanying certainty factors. As with the group membership list, the coordinator at this point is free to modify the recommended list of tools to reflect his own desires.

The module is also implemented in EXSYS Professional and has 30 rules. The list of selected tools is written an ASCII file which is referenced later by the group facilitator.

5.2 Hardware/Software Selection

- **Expert System Tools**

  Early in the project, five well-known expert system shells were chosen for detailed study after an extensive survey of available commercial products including EXSYS, KEE, GOLDWORKS, and TI PERSONAL CONSULTANT PLUS (Other packages were eliminated for a variety of reasons). Neuron Data's Nexpert Object was chosen because it was believed to have all of the features advertised by the others and few of the faults. A short product review of Nexpert Object appears in *Appendix IV*.

  It was discovered that many of the features that appeared attractive were not yet implemented in Nexpert Object. On January 11, 1989, an upgraded version of Nexpert (1.1) was delivered. It appeared to be a substantial improvement over the earlier version, but time was been lost in overcoming the deficiencies of the earlier version.

  The first prototype of ESP used Nexpert Object's NORT (Nexpert Object Run Time) module. Many problems were encountered while using Nexpert and NORT. It was difficult to learn, difficult to use, peformed very slowly, required too much memory, contained too many 'bugs', was not truly global, had a poor interface with other programs, and lacked adequate documentation and support. More details of these problems appear in an earlier report [4].

  Version 1.2 of Nexpert Object was scheduled for release in June 1989 but not received. Better access to classes, objects, and properties was promised, and some of the bugs were to be corrected. Version 2.0 was also underway, but Nexpert has a long way to go before it becomes a useful prototyping environment.

  ESP was finally implemented using EXSYS Professional to overcome the problems with Nexpert Object. A product review of EXSYS Professional apperars in *Appendix V*. No major problems have occurred with the use of this product.

- **Other Software**

Both "V...mont Views" and "Windows for C" were used to build the interface. C was chosen for portability to a Unix environment. Turbo Pascal 4.0 was used to implement the calendar scheduler, and dBase III + was chosen for the database and database management system due to its availability, portability, and ease-of-use.

- **Hardware**

  Appropriate hardware and software were selected and acquired and a subset of the revised design has been implemented. The prototype was developed on an AT&T 6386 personal computer with a 135 MB hard disk, and 4 MB RAM.

  AIRMICS currently uses a mix of IBM compatible PCs, Sun 3/50s, and Sun 386is networked together with a Sun server. The Sun workstations use the UNIX operating system, while the PCs use DOS. The IOIS prototype works in the DOS environment.

6 Conclusion

The only portion of the Automated Session Manager (ASM) currently implemented is the Expert Session Planner (ESP) component. ESP is implemented on an AT&T 6386 WGS workstation. The prototype has met with some initial success in knowledge base validation. The knowledge for the tool selection module was acquired through extensive interviews with local expert facilitators. The knowledge has initially been represented in the form of IF-THEN rules with the possibility of representation in the form of frames in future versions. Since the selection of tools is primarily a design problem, a forward-chaining inferencing strategy is used. Further, 20 user-profiles are stored in the database, implemented in (Dbase III+) and loaded at run time by the prototype system.

The prototype system is implemented using EXSYS Professional in the DOS environment. This environment satisfies many of the implementation criteria by providing graphic interfaces; portability among DOS, Vax, and Unix products; excellent database and knowledge base support; and external program calls.

ESP has been pilot-tested with case studies and field experiments. Initial test results have been positive as the tool has proved to accurately match the expert's prescriptions for group and tool selections. Further enhancements, including intelligent support for the facilitation stage (Expert Session Manager) which uses the output from ESP to automate the facilitation process, are being designed.

In summary, Huber [1] notes that the success of a particular Group Decision Support System depends on the range of tasks supported and its frequency of use (both of which are mutually dependent). Through the added functionality and flexibility of a chauffeured, asynchronous, distributed GDSS, ASM and AGDSS provide a sufficient number of tools and adequate support to achieve the critical threshold necessary for a high frequency of use.

References

[1] Huber, George P., "Issues in the Design of Group Decision Support Systems," *MIS Quarterly*, September 1984, pp. 195-204.

[2] Liu Sheng, O., Motiwalla, M., Nunamaker, J., & Vogel, D., "A Framework to Support Managerial Activities Using Office Information Systems," To appear in *Journal of Management Information Systems*.

[3] Nunamaker, Jay F., Amaravadi, Chandra S., Motiwalla, Luvai F., "OSA: An Office Systems Architecture for Supporting Managerial Activities", *U.S. Army Information Systems Engineering Command's Technology Strategies Conference*, February 1988.

[4] Nunamaker, J., Liu Sheng, O., Aiken, M., Amaravadi, C., Higa, K., Morrison, C., and Motiwalla, L. "Integrated Office Information System (IOIS)," technical report, submitted to the Army Institute of Research in Management Information, Communications and Computer Science (AIRMICS), June 15, 1989.

Appendix I

ESP: A Consultation System for Electronic Meeting Systems

Pre-Session Planning

Milam W. Aiken
Luvai F. Motiwalla
Olivia R. Liu Sheng
Jay F. Nunamaker, Jr.

Department of Management Information Systems
College of Business and Public Administration
University of Arizona 85721
602-621-2748
Bitnet: AIKEN@ARIZVAX
Internet: AIKEN@MIS.ARIZ.EDU

December 6, 1989

Abstract

As group electronic meeting systems (EMS) continually evolve, more powerful methods of managing these systems are needed. Specifically, an efficient and effective means of supporting the pre-session planning processes of human facilitators is required to lead to a successful group meeting. This paper presents a support tool which demonstrates the viability of a rule-based consultation system as a mechanism for effective EMS pre-session planning.

An additional motivation for the development of an on-line consultant for pre-session planning is the examination of the human facilitator's mental model of the planning process. Through the successive steps of interviewing experts, prototype development, and prototype validation, much can be learned about the nature of facilitator interaction with the group and the EMS.

This paper presents a background investigation of the problems of supporting EMS and then outlines a detailed description of the architecture of Expert Session Planner (ESP), a prototype solution to many of these problems. A case study involving a typical office organization illustrates the use of this tool, and validation study results indicate several interesting characteristics of human/computer interaction. Finally, research directions are described for the further integration of expert system technology with EMS.

Keywords: Expert systems, group decision support systems, electronic meeting systems, artificial intelligence

1 Introduction

1.1 A Definition of the Problem Environment

Information technology has received widespread interest in recent years for the support of group productivity [Huber 1984, Keen and Scott Morton 1978, Richman 1987, Straub and Beauclair 1988]. Previous research has shown the benefits of information technology in electronic meeting environments [Quinn et al 1985, Steeb and Johnston 1981, Turoff and Hiltz 1982, Watson 1987, Zigurs 1987], and reports of time and cost savings of up to 56 percent are prevalent in the literature [Gallupe 1985, Lewis 1982, Nunamaker et al 1989]. In addition, these group systems have been shown to foster collaboration, communication, deliberation, and negotiation [Apple et al 1986, Beauclair 1987, Easton 1988, Gray 1981, Kull 1982].

Early computer-based systems for group support, described as Group Decision Support Systems (GDSS), included *"... a set of software, hardware, language components, and procedures that support a group of people in a decision related meeting"* [Huber 1984]. However, there is some disagreement about what exactly constitutes a GDSS [Baskin et al 1988, Kraemer and King 1986]. Although some definitions of GDSS include simple conferencing systems or special purpose systems [Hale and Haseman 1987, Kerr and Hiltz 1982, Turoff et al 1988], these categorizations are too limited. A broader description of information technology to support group meetings can be stated as Electronic Meeting Systems (EMS): [Dennis et al 1988]:

> *An information technology-based environment that supports group meetings, which may be distributed geographically and temporally. The information technology environment includes, but is not limited to, distributed facilities, computer hardware and software, audio and video technologies, facilitation, and applicable group data. Group tasks include, but are not limited to, communication, planning, idea generation, problem solving, issue discussion, negotiation, conflict resolution, systems analysis and design, and collaborative group activities such as document preparation and sharing.*

1.2 EMS Problem Areas

Most early systems to support group meetings met with only limited success [Kraemer and King 1986]. Although many causes may be cited for their failure, a partial explanation can be found in the lack of support given to these systems.

For example, little is known about EMS facilitation. There are many broad questions on the effect of group and individual characteristics, the task, the context, and technological limitations on EMS processes

and outcomes [Nunamaker et al 1989]. These questions may be best framed in light of the stages followed in a typical EMS:

1. **Pre-Session Planning**

   During this stage, a session facilitator talks with a representative from the organization which needs to meet to resolve a problem. This representative provides the facilitator with information regarding the group characteristics and the task faced. Using this information, the facilitator can construct an agenda of EMS tools and meeting times to bring about a successful outcome. In addition, the facilitator can help the representative come up with a list of appropriate group participants from the organization to effectively address all issues of the meeting.

   Selecting the correct tools and session participants becomes increasingly important as EMS evolve to distributed systems in which a facilitator is not physically present [Nagao et al 1989]. EMS support is needed to resolve the uncertainty involved in making these group and tool selections.

2. **Session Facilitation**

   In this stage, the facilitator follows the agenda prepared in the planning stage and leads the group through the session using appropriate techniques for the task at hand. Some EMS techniques include: generation of ideas or plans, organization of ideas, consolidation and focusing of ideas, generation of proposals, and assessment of proposals [DeSanctis and Gallupe 1987]. These techniques can be applied to decision types described as Planning (generation of action-oriented plans), Creativity (generation of novel ideas), Intellective (selection of the correct alternative), Preference (selection of an alternative for which there is no correct answer), Cognitive Conflict (resolution of conflicting viewpoints), and Mixed Motive (resolution of conflicting motives or interests).

   Much EMS software has been written to facilitate group processes. As an example, the University of Arizona's Collaborative Management Room provides a suite of tools known as GroupSystems which includes: Electronic Brainstorming (EBS), Automated Delphi, Nominal Group Technique (NGT), Issue Analyzer, Issue Organizer, Policy Formation, Vote Selection, Alternative Evaluator, Topic Commenter, Stakeholder Identification and Assumption Surfacing (SIAS), and Enterprise Analyzer.

   During the facilitation stage of an EMS session, the group facilitator must monitor the on-going discussion to help lead it in appropriate paths of discourse and to terminate the session as comments become redundant or superfluous. As groups grow large, facilitators may be overwhelmed by the

sheer volume of information. EMS support is needed to ease the burden of facilitation.

In addition to facilitators, group members may need assistance using an EMS. An organization EMS may be only infrequently used, resulting in less familiarity with the system and even less use [Huber 1984]. EMS support in the form of on-line tutorials, intelligent help, and other functions may be needed to keep users afresh.

3. **Post-Session Analysis**

At the end of the session, the facilitator meets with the group representative again to ensure that the session effectively met the group's needs. The post-session analysis includes summary reports of decisions that were made during the meeting as well as a complete transcript of comments generated by the group participants. This output can serve as input to other software such as CASE tools, databases, etc. EMS support is needed here to impose structure on this output.

1.3 The Expert System Solution

Artificial intelligence may be the most important technical contributor to the future of EMS [Johansen 1988]. Expert systems (ES) may transform EMS from passive agents that process and present information to active agents that enhance interactions [Ellis et al 1988]. The goal is to design such integrated systems in a way that enhances desirable procedural and social group processes. This transformation to "softer software" [Johansen 1988] includes the integration of expertise to simplify use and operation of EMS.

The expert system (ES) paradigm has found many successful commercial applications in a variety of areas [Michaelson and Michie 1983, Winston and Prendergast 1984], and the expert system development methodology continues to be studiously researched as a promising frontier of artificial intelligence. Turban and Watkins [Turban and Watkins 1986] have reported a framework for applying expert systems to decision support systems (DSS), a positive step in the direction of evoking more powerful management information systems. The next logical step is to apply ES technology to GDSS or EMS *(Figure 1)* [Aiken, Liu Sheng, and Vogel 1989].

DeSanctis and Gallupe argue that research should begin with a study of Level 1 GDSS/EMS systems (systems which provide a communication medium only) before proceeding to higher level systems (see [DeSanctis and Gallupe 1987] for a thorough explanation of GDSS levels). Many current EMS tools provide Level 1 communication media and Level 2 decision modelling techniques and have been reviewed thoroughly by the literature; an integration of expert systems with EMS will provide Level 3 functionality including the automated rule-based generation of procedures for EMS sessions.

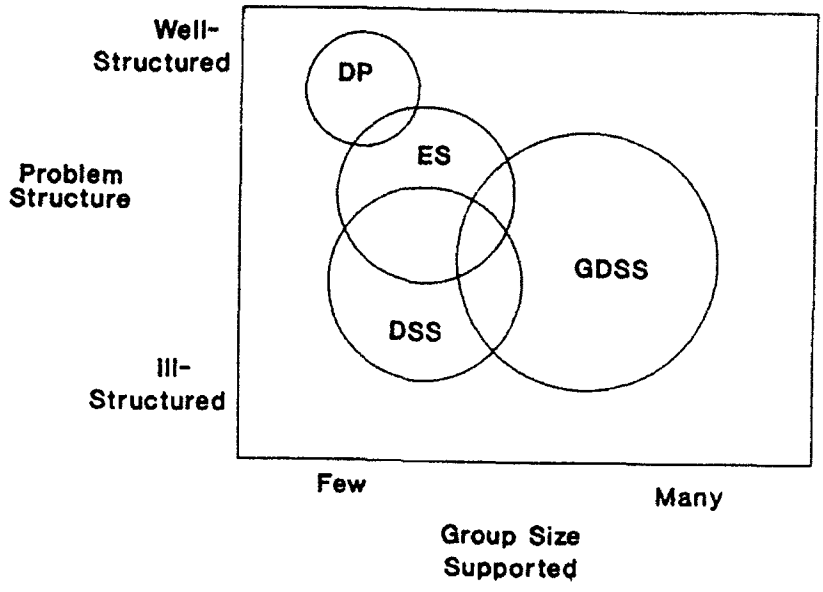


Figure 1: Evolving to ES/GDSS

Intelligent support can be applied to solve various problems in EMS including information retrieval, information analysis, and session faciliation among others [Agarwal and Prasad 1989,Jarke 1986]. This paper, however, focuses on applying ES to the pre-session planning stage of an EMS session. Expert assistance at this stage can alleviate the burden placed upon a human facilitator, enhance the quality and outcome of the ensuing session, and distribute the scarce expertise of EMS pre-session planning [Aiken et al 1989, Aiken et al 1990]. Effective intelligent support for pre-session planning is becoming even more crucial to computer-based meetings as the trend toward distributed EMS grows (due in part to the cost and time advantages over its face-to-face counterpart).

Expert Session Planner (ESP) is a prototype ES designed to support EMS session facilitators during critical pre-session planning. ESP allows scarce human facilitator knowledge to be distributed and provides consistency across sessions. In addition, ESP provides training for novice session facilitators in the selection of EMS software. Finally, this tool offers a foundation upon which to build further intelligent support software for group meeting processes.

1.4 Outline of the Paper

The remainder of the paper is organized as follows. Section two describes the architecture of the tool, and section three presents results from validation studies. Section four summarizes the paper and lists

future research directions.

2 Description of the Prototype Design and Architecture

2.1 Prototype Design

Expert Session Planner (ESP) is designed specifically to assist EMS facilitators in the pre-session planning stages of group meetings. ESP can assist the facilitator in the selection of appropriate tools and participants for the session. In addition, ESP can scan the calendars of the session participants to determine convenient times when face-to-face meetings are needed. ESP thus allows a meeting coordinator with minimal knowledge of the EMS tools and the processes involved in pre-session facilitation to plan and coordinate electronic meetings. The following subsections discuss knowledge acquisition, knowledge representation, and the inferencing strategy employed in the design of ESP.

2.1.1 Knowledge Acquisition

Knowledge acquisition is perhaps the most difficult and tedious phase of any ES project. Effective knowledge acquisition strategies must be employed for the ultimate success of the system. Designers (or knowledge engineers) should ideally already be at least familiar with the basics of the problem at hand; if not, they should review applicable textbooks and references before approaching the human expert. Only by establishing a mental model framework will the designers be able to file the anecdotes and rules-of-thumb acquired from interviewing the expert. Also, a minimum set of knowledge in any area is a prerequisite for posing incisive questions.

For the prototype ESP explicated in this paper, the authors met over a period of three months with a panel of local EMS experts to elicit the factors involved in the tool selection process. This panel of experts included group facilitators with several years of EMS session experience as well as tool developers and researchers. The authors had moderate experience with EMS terminology and methodology through prior study. In the final stages of knowledge acquisition, an EMS Issue Analysis and Organization session was conducted in the University of Arizona's Collaborative Manangement Room to categorize the factors involved. In addition, the authors met with Major Ted Hengst at the Army Institute of Research in Management Information, Communications, and Computer Science (AIRMICS) office to determine the tools and participants needed for meetings based on job responsibilities, work interests, and departmental affiliations. This knowledge was incorporated into the system and is discussed in more detail below.

2.1.2 Knowledge Representation

Once the underlying knowledge of the expert task has been acquired, a theoretical representation of the knowledge must be developed. Some representations include: IF-THEN rules (perhaps the simplest and most prevalent technique), Frames and Objects (popular for their powerful inheritance and hierarchical properties), and Semantic Networks (noted for their flexible, yet powerful structures). The choice of knowledge representation is perhaps the guiding force of the ultimate implementation.

An ES for pre-session planning may benefit from a frame or object representation since group and tool characteristics often have hierarchical structures. For example, group participants can be classified by organizational affiliation, interests, or responsibilities. An ES which selects appropriate group participants can rely on certain inherited features instead of features which are expressed for each group member. Likewise, EMS tools can be classified by their applicabilty for a particular group or problem. Inheritance of these characteristics may allow the ES to procede more efficiently in determining which EMS tools to select.

The initial prototype of ESP for tool selection uses IF-THEN rules. The group selection module was designed with the procedures of selecting the participants represented with IF-THEN rules and the facts about the participants stored in a database. This reduces the number of rules required in selecting the participants. However, the knowledge base can be further refined with the use of more powerful representational techniques such as semantic nets, frames, and object-oriented approaches which will be investigated in the future.

2.1.3 Inferencing Strategy

The expert task must also be analyzed as to the fundamental nature of the problem: one of design, diagnosis, or both. Design problems typically feature a relatively limited set of underlying conditions and a much greater number of goals; diagnosis problems include relatively few goals with many conditions which must be verified. For example, the problem of group selection is one of design since the number of potential participants is likely to be much greater than the number of criteria involved in the group configuration. Tool selection, on the other hand, is primarily a matter of diagnosis; there is a relatively limited number of tools while there is a much greater number of factors involved in their selection. A forward-chaining inferencing strategy is most efficient with design problems while a backward-chaining strategy is most economical with diagnosis tasks. Some problems may involve diagnosis and design, indicating a need for a hybrid inferencing strategy.

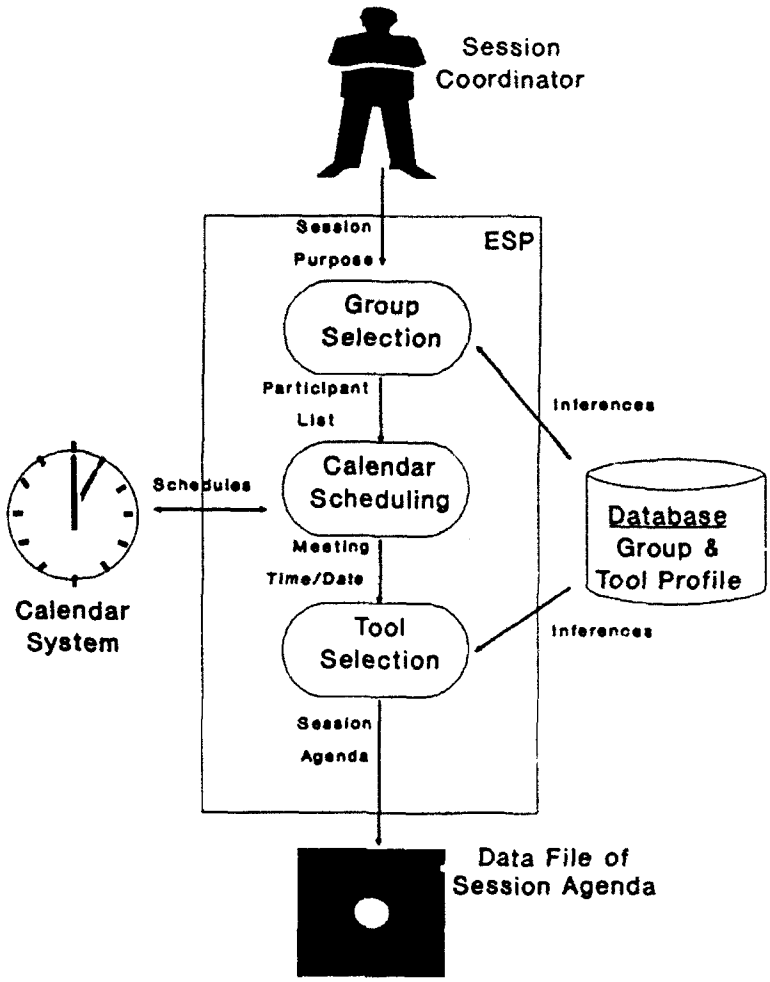


Figure 2: Architecture of ESP

For tool selection, the ESP prototype uses an exhaustive, backward-chaining inferencing strategy with provisions for rule uncertainty (to resolve conflicting rules). For group selection, ESP uses forward-chaining.

2.2 ESP: Architecture

Figure 2 shows the high-level architecture of the prototype system currently being developed at the University of Arizona's Collaborative Management Room. The system consists of three principle components: the group selection, calendar scheduling, and tool selection modules.

1. **Group Selection Module**

   The facilitator or meeting coordinator begins ESP with the group selection module called to determine the appropriate participants for the meeting *[Figure 3]*. The type of the meeting, the topic of the meeting, and personnel interests, responsibilities, and organizational affiliations are all factors used by the knowledge base to configure a query for a search through a user profile database.

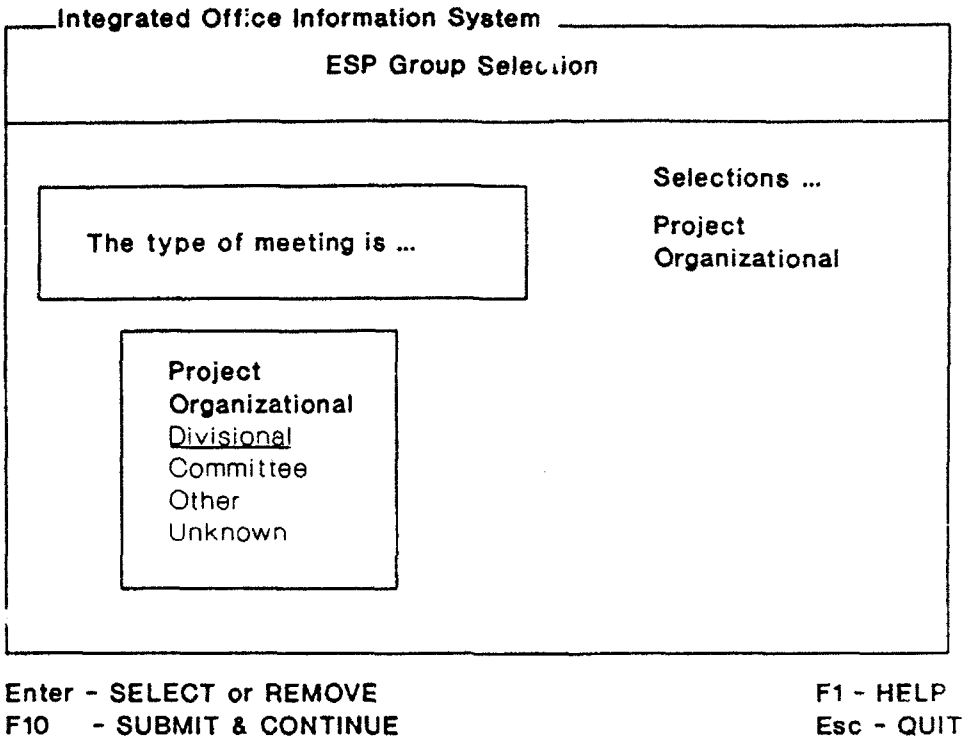


Figure 3: ESP Group Selection Sample Screen

A DBMS takes this query to generate a list of participants for the EMS session. The meeting coordinator can use the list as is or make further changes depending on his preferences. The role of group selection is merely to provide a starting point for the meeting coordinator. This group selection module arose out of prior research on intelligent mail systems [Higa et al 1988, Motiwalla 1989, Motiwalla et al 1989].

The meeting topic helps in determining the first-cut list. For instance, if the topic involves discussion of a project, then all members responsible for the project are selected. Further, the supervisor of the project and any member who has listed an interest in the area of the project will be selected. The meeting type, on the other hand, helps in determining who should be added or deleted from the list. For instance, if it is a preliminary discussion meeting, then the supervisor will be removed, or if the discussion involves financial matters, then the financial officer involved with the project will be added.

The group determination module is implemented using EXSYS Professional (a product of EXSYS, Inc.). The knowledge base consists of 39 rules which configure a query for a DBase III+ DBMS. This DBMS parses the query, searches through a DBase III+ database of 37 user-profiles, and yields a session participant list. This list is then written to an ASCII file for reference by the facilitator.

2. **Calendar Scheduling Module**

```
The meeting could be at:
   7:30 to 10:30, Thur, 5-18-1989
Show another possible time (Y/N)? Y
The meeting could be at:
   9:00 to 12:00, Thur, 5-18-1989
Show another possible time (Y/N)? N
Schedule the meeting at this time (Y/N)? Y
```

Figure 4: ESP Group Scheduling Sample Screen

Once the participant list is determined, the calendar scheduling module is used to determine times which are convenient for all session participants *[Figure 4]*. In addition, the module checks the calendar information of the EMS meeting room to see if it can be scheduled when the participants can meet. If all calendars match, the scheduling module makes the appropriate appointments in the calendars of all concerned. If no time can be found that is convenient for all participants or the meeting room cannot be scheduled at this time, a distributed, asynchronous EMS tool will need to be used (to be determined in the final stage of *ESP*).

The scheduling module is currently implemented in Turbo Pascal although the tool will be ported to a Unix C environment along with most of the EMS tools in the near future.

3. Tool Selection Module

Once a group membership list has been constructed and meeting times have been arranged, the meeting coordinator uses the tool selection module to determine the EMS tools required for the session. The coordinator must have already used the group selection module at this point because he is expected to answer questions regarding the group's characteristics. For example, the coordinator must know the extent of the participants' familiarity with the topic as well as the extent of common knowledge among the group members *[Figure 5]*. Additional questions are asked concerning problem characteristics such as whether or not the problem can be segmented and whether or not stakeholder identification is important. Once all of the questions are answered, a list of recommended tools is written to a file with their accompanying certainty factors. As with the group membership list, the coordinator at this point is free to modify the recommended list of tools to reflect his own desires.

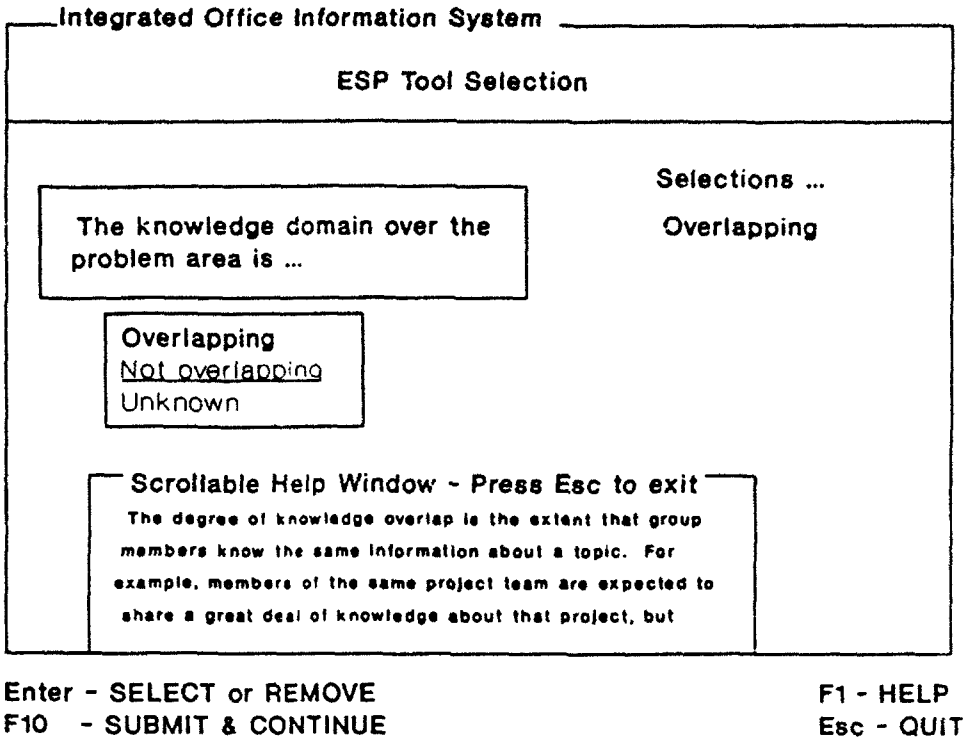


Figure 5: ESP Tool Selection Sample Screen

This module is also implemented in EXSYS Professional and has 68 rules. The list of selected tools is written to an ASCII file which is referenced later by the group facilitator.

2.3 ESP Implementation Status

ESP is implemented on an AT&T 6386 WGS workstation with two megabytes of main memory and an 80-megabyte hard disk. As was noted above, the prototype was developed using EXSYS Professional, Turbo Pascal, and DBase III+ under an MS-DOS environment. EXSYS Professional was chosen for its graphic interfaces; portability among DOS, Vax, and Unix products; excellent database and knowledge base support, and external program calls.

In summary, Huber [Huber 1984] notes that the success of a particular EMS depends on the range of tasks supported and its frequency of use (both of which are mutually dependent). Through the added functionality and flexibility of a chauffeured, asynchronous, distributed EMS, ESP provides a sufficient number of tools and adequate support to achieve the critical threshold necessary for a high frequency of use.

Huber also states that effective facilitation depends upon the technical competence of the users of an EMS and knowledge of the planning process. As far as technical competence is concerned, the facilitators and group participants are assumed to already be familiar with each of the EMS tools in the agenda.

ESP adds to the technical competence by providing the most appropriate tool for a given task and group. The most important contribution of ESP, however, is the addition and dissemination of knowledge about the planning process. The prototype satisfies this requirement through accumulated knowledge of the requirements, purposes, and goals of the group to be facilitated. The type of organization, its methods and goals, and its purposes for the session are all important for the facilitator (human or automated-chauffeur) to know. Knowledge about the initiator and members participating is also important. The prototype system meets all of these requirements.

3 A Scenario Demonstrating the use of ESP

A case study was conducted using information from the Army AIRMICS office to demonstrate the feasibility of ESP. The scenario involved project funding, a task found in practically all organizational offices which deal with budgeting. Some details and names in this scenario have been changed for confidentiality.

The AIRMICS staff meets annually to decide upon projects which will be funded in the upcoming fiscal year. The chief of AIRMICS, Mr. Smith, decides that a meeting needs to be called. As the director of AIRMICS, he assumes the task of group coordinator and initiates the use of ESP to set up the EMS session. Mr. Smith brings up ESP on his networked microcomputer system.

3.1 Group Selection

The first task to accomplish is the selection of the group participants. Mr. Smith answers a series of questions regarding the meeting type and meeting topic using ESP's group selection module. From the organization's personnel profile database, ESP determines that Mr. Jones, Mr. Johnson, and Mr. Adams need to attend because of their responsibity for project funding. In addition, Mr. Jefferson and Mr. Madison are included based upon their interest in project planning.

3.2 Group Scheduling

Mr. Smith then uses the Group Scheduling Module of ESP to determine a meeting time convenient for all participants. From the individual calendars of all group participants, ESP determines that May 18 at 9:00 am is open and automatically blocks three hours for a EMS session in all of the calendars. In addition, the scheduling module determines that the EMS decision room is available at this time and schedules it accordingly.

Parameter	Value
Knowledge Overlap	Much
Group Familiarity	High
Need for Consensus	Low
Need to Educate	Low
Problem Segmentation	Yes
Group Experience	High
Management Style	Hierarchical
Idea Divergence	Wide
Policy Needed	Yes
Idea Consolidation	Yes
Voting Criteria	Many
Voting Nature	Judgmental
Stakeholder ID	Needed

Table 1: AIRMICS Scenario Knowledge

3.3 Tool Selection

ESP then asks the group coordinator a series of questions regarding the problem and group. Since all participants selected by the group selection module have similar backgrounds and have met several times before, he states that there is much overlap of knowledge over the problem area. Similarly, he states that there is high familiarity with the topic. Table 1 presents the remainder of his ~~the~~ answers. Table 2 presents the recommended tool list. These tools are then used during the following facilitation stage of the EMS session.

4 ESP Validation

Preliminary results from four field studies and ten historical case studies have revealed numerous insights into the pre-session planning process.

Tool Category	Tool Name	CF
Idea Generation Tool	Electronic Brainstorming	.76
Issue Organization Tool	Issue Organizer	.65
Voting Tool	Vote Selection	.75
Stakeholder Tool	Stakeholder Identification and Assumption Surfacing	.82

Table 2: Recommended Tools for AIRMICS Scenario

1. Very experienced facilitators expressed little interest in having an on-line tool (such as ESP) for assistance during planning, although relatively inexperienced facilitators expressed interest. This result was anticipated and provides an additional motivation for the design of a consultation and tutoring tool for novices.

2. Facilitators may not be using the tools that are best for a given group, task, and environment. Even though there is a justification for using certain tools, some are rarely if ever used. For example, Policy Formation and Nominal Group Technique often are highly ranked, but facilitators never use them.

   The Nominal Group Technique probably should be used more than it has been. The reason seems to be "familiarity breeds comfort" as much as anything. In addition, facilitators often find it a little more troublesome to use the Nominal Group Technique as compared to starting a group cold on Electronic Brainstorming.

   As for, Policy Formation, the tool is rarely used, which probably makes facilitators leery of using the tool (which works fine) through lack of experience.

3. A comparison of ESP's recommendations based on facilitator responses to its inquiries and the tools actually used showed a match of about 90 percent. This is encouraging for an early prototype.

4. Perhaps a 100 percent match will never be achieved. This is due to:

   - Facilitators may remain with tools more familiar to them. The match would be higher than 90 percent if the problem with # 2 above were resolved. For example, one rule asks: Is a policy statement needed? The facilitator responded YES and ESP recommended Policy Formation. It wasn't used, however.

- Facilitators sometimes make mistakes. For example, a facilitator said that one-factor voting was not needed, and yet, rank-order voting using one factor was then used. Perhaps there were some hidden rules or a hidden agenda involved in this decision.

Additional validation of ESP is being conducted through approximately 70 historical case studies and on-going field studies. This research will lead to further insights of the facilitator's mental model of the pre-session planning process.

5 Conclusion

An ES can make an EMS a more flexible and powerful aid in group decision processes. Existing EMS incorporating a wide variety of tools are limited by the scarce expertise of human facilitators; integrating ES with EMS will enable the replication of this scarce EMS session management knowledge and will allow remote, distributed session planning to be conducted.

Scott Morton has noted that conventional DSS may be supplanted by EDSS (expert decision support systems) [Scott Morton 1984]; the same may be said for GDSS and EMS. ES will some day monitor a group's decision processes, analyze the content of the discussion, and direct the group on alternate paths to reach a desired goal.

Humans will never completely disappear, however, as there may continue to be a need for "hand-holding," the reassuring presence of a human expert to extract group participants from difficult quagmires and imbroglios. Also, for small groups with a limited need for a vast array of EMS tools, a human facilitator may still be the most cost-effective means of controlling group collaborative work.

Potential developers of an integrated ES/EMS must keep in mind the many limitations of the respective technologies to assure a successful outcome. Particularly, additional research is required in the area of deep knowledge, the underlying theory of group processes, to adequately represent the human behavior emulated by an ES group facilitator. ESP presents a panoply of rich research areas including technical, behavioral, and design issues which are as yet relatively unexplored. Work is currently under way to provide intelligent support for the facilitation and post-session analysis stages of EMS sessions through natural language processing and additional ES features. In addition, more rules are being added to the tool selection module to incorporate knowledge about the nascent field of asynchronous, distributed EMS.

Acknowledgments

This project is supported by a grant from the Army Institute of Research in Management Information, Communications, and Computer Science (AIRMICS), Altanta, GA Grant #: DAKF-11-88-C-0021. We would like to thank Dr. Kunihigo Higa, Richard Orwig, Pamela Howard, Hsiao-Yu Wu, and Chih-Ping Wei for their work on the ESP prototype.

References

1. Agarwal, R. and Prasad, K., 1989, "Enhancing the Group Decision Making Process: An Intelligent Systems Architecture," *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences*, Vol. III, pp. 273-279.

2. Aiken, M., Liu Sheng, O., and Vogel, D., 1989, "Integrating Expert Systems with Group Decision Support Systems," working paper, University of Arizona.

3. Aiken, M., Motiwalla, L., Liu Sheng, O., and Nunamaker, J., 1989, "An Expert System Approach to Group Decision Support System Tool Selection," *Proceedings of the 1989 Annual National Conference of the Association of Computer Educators*, pp. 96-104.

4. Aiken, M., Motiwalla, L., Liu Sheng, O., and Nunamaker, J., 1990, "An Expert System for Pre-Session Group Decision Support Systems Planning," *Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences.*

5. Applegate, L. M. et al, 1986, "A Group Decision Support System for Idea Generation and Issue Analysis in Organizational Planning," *Proceedings of the Conference on Computer-Supported Cooperative Work*, pp. 16-34.

6. Baskin, A. B. et al, 1988, "Integrated Design as a Cooperative Problem Solving Acitivity," working paper, Univerisity of Illinois.

7. Beauclair, R. A., 1987, "An Experimental Study of the Effects of GDSS Process Support Applications on Small Group Decision Making," Unpublished Ph.D. dissertation, Indiana University.

8. Dennis, A., George, J., Jessup, L., Nunamaker, J., and Vogel, D., 1988, "Information Technology to Support Electronic Meetings," *MIS Quarterly*, Vol. 12, No. 4, pp. 591-624.

9. DeSanctis, G. and Gallupe, R., 1987, "A Foundation for the Study of Group Decision Support Systems," *Management Science*, Vol. 33, No. 5.

10. Easton, A., 1988, "An Experimental Study of Automated Versus Manual Support for Stakeholder Identification and Assumption Surfacing with Small Groups," Unpublished Ph. D. dissertation, University of Arizona.

11. Ellis, C., Gibbs, S., and Rein, G., 1988, "Groupware: The Research and Development Issues," *Report STP-414-88*, MCC Software Technology Program, Austin, TX.

12. Gallupe, R. B., 1985, "The Impact of Task Difficulty on the Use of a Group Decision Support System," Unpublished Ph. D. dissertation, University of Minnesota.

13. Gray, P. et al, 1981, "The SMU Decision Room Project," *Transactions of the First International Conference on Decision Support Systems*, pp. 122-129.

14. Hale and Haseman, 1987, "EECS: A Prototype Distributed Executive Communication and Support System," *Proceedings of the Twentieth Hawaii International Conference on System Sciences*, Vol. 1, pp. 557-565.

15. Higa, K. et al, 1988, "A Theoretical View and a Design Approach to a Knowledge-Based Mail System," Working Paper, University of Arizona.

16. Huber, G., 1984, "Issues in the Design of Group Decision Support Systems," *MIS Quarterly*, Vol. 8, No. 3, pp. 195-204.

17. Jarke, M., 1986, "Knowledge Sharing and Negotiation Support in Multiperson Group Decision Support Systems," *Decision Support Systems*, No. 2, pp. 93-102.

18. Johansen, R., 1988, *Groupware: Computer Support for Business Teams*, Free Press, New York.

19. Keen, P. and Scott Morton, M., 1978, *Decision Support Systems: An Organization Perspective*, Addison-Wesley, Reading, MA.

20. Kerr, E. and Hiltz, S., 1982, *Computer-Mediated Communication Systems*, Academic Press, New York.

21. Kraemer, K. and King, J., 1986, "Computer-Based Systems for Cooperative Work and Group Decision Making: Status of Use and Problems in Development," *Proceedings of the 1986 Conference on Computer Supported and Collaborative Work*, pp. 353-375.

22. Kull, D., 1982, "Group Decisions: Can a Computer Help?" *Computer Decisions*, Vol. 14, No. 5, pp. 70-84.

23. Lewis, F., 1982, "Facilitator: A Microcomputer Decision Support System for Small Groups," Unpublished Ph. D. dissertation, University of Louisville.

24. Michaelson, R. and Michie, D., 1983, "Expert Systems in Business," *Datamation*, Vol. 29, No. 11, pp. 240-246.

25. Motiwalla, L. 1989, "A Knowledge-Based Messaging System: Framework, Design, Prototype Development, and Validation," Unpublished Ph.D. Dissertation, University of Arizona.

26. Motiwalla, L. et al, 1989, "A Framework to Support Managerial Activities Through Knowledge-based Electronic Mail Systems," *Proceedings of the Twenty-Second Hawaii International Conference on System Sciences.*

27. Nagao, D., Parsons, C., Herold, D., 1989, "Group Dyamics in Distributed Computer Supported Groups," working paper, College of Management, Georgia Institute of Technology.

28. Nunamaker, J., Vogel, D., Heminger, A., Martz, B., Grohowski, R., and McGoff, C., 1989, "Experiences at IBM with Group Support Systems: A Field Study," *International Journal of Decision Support Systems*, Vol. 5, No. 2, pp. 183-196.

29. Quinn, R. et al, 1985, "Automated Decision Conferencing. How it Works," *Personnel*, Vol. 62, No. 11, pp. 49-55.

30. Richman, L., 1987, "Software Catches the Team Spirit," *Fortune*, Vol. 115, No. 12.

31. Scott Morton, M., 1984, "Expert Decision Support Systems," a paper presented at the special DSS conference, Planning Executive Institute and Information Technology Institute, New York, New York, p. 21.

32. Steeb, R. and Johnston, S., 1981, "A Computer-Based Interactive System for Group Decision Making," *IEEE Transactions on Systems, Man, and Cybernetics*, Volume SMC-11, Number 8, pp. 544-552.

33. Straub, D. and Beauclair, R., 1988, "Current and Future Uses of Group Decision Support System Technology: Report on A Recent Empirical Study," *Journal of Management Information Systems*, Vol. 5, No. 1, pp. 101-116.

34. Turban, E. and Watkins, P., 1986, "Integrating Expert Systems with Decision Support Systems," *MIS Quarterly*, Vol. 10, No. 2, pp. 121-134.

35. Turoff, M. et al, 1988, "The TEIES Design and Objectives: Computer Mediated Communications and Tailorability," Working paper, Computerized Conferencing and Communications Center, New Jersey Institute of Technology, Newark, NJ.

36. Turoff, M. and Hiltz, S., 1982, "Computer Support for Group Versus Individual Decisions," *IEEE Transactions on Communications*, Vol. 30, No. 1, pp. 82-90.

37. Watson, R., 1987, "A Study of Group Decision Support System Use in Three and Four-person Groups for a Preference Allocation Decision," Unpublished Ph. D. dissertation, University of Minnesota.

38. Winston, P. and Prendergast, K. (eds.), 1984, *The AI Business: The Commercial Uses of AI*, MIT Press, Cambridge, MA.

39. Zigurs, I., 1987, "The Effect of Computer-Based Support on Influence Attempts and Patterns in Small Group Decision-Making," Unpublished Ph. D. dissertation, University of Minnesota.

About the Authors

Milam W. Aiken received the B.S. degree in Engineering and the Master's of Business Administration degree from the University of Oklahoma and the B.A. degree in Computer Science and the B.S. degree in Business from the State University of New York. He is currently a Ph.D. student in Business Administration with a major in MIS at the University of Arizona, and his research interests include office automation, expert systems, and group decision support systems.

Luvai F. Motiwalla received his B.COM. degree from University of Bombay, B.B.A. degree in Business Administration from the Penn State University and M.S. and Ph.D. in MIS from the University of Arizona. He is currently an assistant professor of MIS at the University of Hartford, West Hartford, Connecticut. His research interests include integration of expert systems technology into the office automation area and studying its impact on managers and knowledge workers.

Olivia R. Liu Sheng received the B.S degree from the National Chiao Tung University in Taiwan, R.O.C. and the Master's and Ph.D. degrees in Business Adminsitration with a major in Computers and Information Systems from the William E. Simon Graduate School of Business Adminsitration, University

of Rochester, Rochester, N.Y. She is an assistant professor of MIS at the University of Arizona. Her principal research interests are analysis and design of distributed information systems.

Jay F. Nunamaker, Jr. received a B.S. degree from Carnegie-Mellon University, B.S. and M.S. degrees in Mechanical and Systems Engineering from the University of Pittsburgh, and a Ph.D. in Systems Engineering and Operations Research at Case Institute of Technology. He is a professor of MIS and Computer Science at the University of Arizona. He is an author of more than 35 papers on the automation of software construction, performance evaluation of computer systems, and decision support systems for system analysis and design and has lectured throughout Europe, Russia, and South America.

Appendix II

Integrating Expert Systems with Group Decision Support Systems

Milam W. Aiken

Olivia R. Liu Sheng

Douglas R. Vogel

Department of Management Information Systems

College of Business and Public Administration

University of Arizona

Tucson, AZ 85721 602-621-2748

Bitnet: AIKEN @ARIZMIS

Internet: AIKEN @MIS.ARIZONA.EDU

March 15, 1990

Abstract

Expert systems are powerful tools serving as adjuncts to decision making and have found wide applicability in a variety of areas. Synthesizing expert systems with group decision support systems has the potential to enhance the quality and efficiency of group communication, negotiation, and collaborative work. This paper examines possible synergies between the two technologies and provides a survey of current partially-integrated systems. Finally, a prototype design of a highly-integrated system is described with directions for further research.

Keywords: Expert Systems, Knowledge-Based Systems, Group Decision Support Systems, Artificial Intelligence

Integrating Expert Systems with Group Decision Support Systems

1 Introduction

Information technology has received widespread interest in recent years for the support of group productivity [33, 38, 60, 68]. Early computer-based systems for group support, described as Group Decision Support Systems (GDSSs), included *"... a set of software, hardware, language components, and procedures that support a group of people in a decision related meeting"* [33] while DeSanctis and Gallupe [17] defined GDSSs as integrated computer-based systems which facilitate solution of semi- or unstructured problems by a group that has joint responsibility for making the decision. There continues to be some disagreement about what exactly constitutes a GDSS [8, 26, 39, 73, 75, 77], although Kraemer and King [43] have noted that GDSSs have evolved beyond their original emphasis on decision making. The term Electronic Meeting System (EMS) has been proposed as a broader definition of information technology to support group meetings. An EMS is [15]:

> *An information technology-based environment that supports group meetings, which may be distributed geographically and temporally. The information technology environment includes, but is not limited to, distributed facilities, computer hardware and software, audio and video technologies, facilitation, and applicable group data. Group tasks include, but are not limited to, communication, planning, idea generation, problem solving, issue discussion, negotiation, conflict resolution, systems analysis and design, and collaborative group activities such as document preparation and sharing.*

For the purposes of this paper, however, the term GDSS will be used to describe this more comprehensive view of group support systems.

Previous research has shown the benefits of information technology in electronic meeting environments [58, 64, 73, 78, 81], and reports of time and cost savings of up to 56 percent have been cited in the literature [23, 46, 56]. In addition, GDSSs have been shown to foster collaboration, communication, deliberation, and negotiation [7, 9, 19, 25, 45]. However, several technological, sociological, and psychological problems still remain with the use of current GDSSs. These problems are manifested in difficulties with user interfaces, group communication, use of GDSS tools, and access to supporting information, among others, and are discussed in more detail in section 2.

The objective of this paper is to describe a systems modelling and research framework for the integration of expert systems (ESs) with GDSSs to alleviate these problems in group support systems. First, a background investigation of some problems with GDSSs is presented in section 2. Section 3 describes seven variations of integrating ESs with GDSSs to address these problems, and section 4 provides a sample highly-integrated ES/GDSS prototype system. Conclusions and directions for further research appear in section 5.

2 Problem Background and Potential Solution

2.1 GDSS Problem Areas

Most early systems to support group meetings met with only limited success [43]. Although many causes may be cited for their failure, a partial explanation can be found in the lack of support given to these systems [32, 76]. Later, more general-purpose systems were developed to more adequately support group collaborative work and communication. An example of such a general-purpose GDSS is found in the Collaborative Management Workshop at the Department of Management Information Systems, University of Arizona. The Workshop consists of twenty-four terminals connected to a file server in a decision room environment with a suite of GDSS tools known as GroupSystems [1, 15].

GroupSystems was designed with an engineering perspective of creating tools that people in decision-making situations might find useful [43]. Further, the system was designed with the rational model of user behavior in mind. In this model, decision makers try to optimize their decisions by gathering a wide variety of information about the problem, developing alternative solutions to the problem, and finally making a choice (what Simon refers to as Intelligence, Design, and Choice [62]). As a consequence of this rational, technology-driven approach to the design of GDSSs, many theoretical questions arise on the effect of group and individual characteristics, the task, the context, and technological limitations on GDSS processes and outcomes *(Figure 1)* [41, 51, 56]. With this research framework in mind, a review of the literature and experience with over 100 groups using GroupSystems at the University of Arizona suggest many problems with current GDSSs and several possible solutions:

1. **Group members need to be encouraged to participate in sessions.**

   In current face-to-face GDSS sessions, group facilitators are able to visually monitor session participants and can encourage individuals to contribute more frequently if necessary. Although such monitoring is not always desirable, a need to monitor session activity becomes even more important

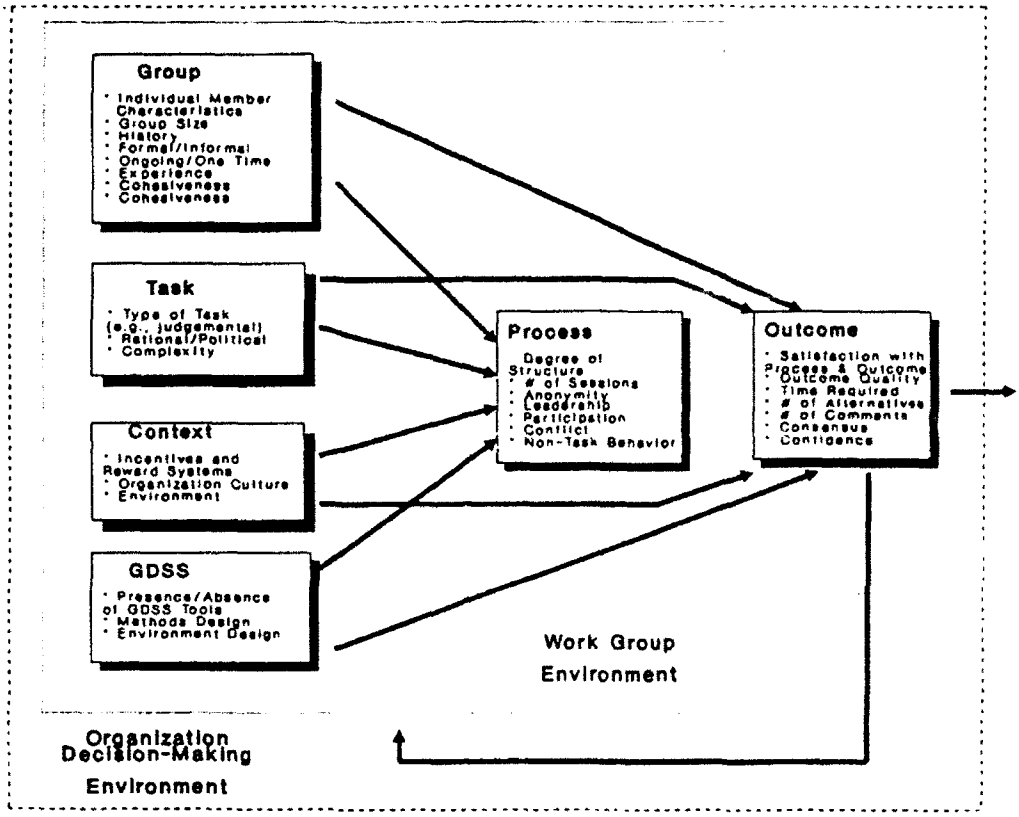


Figure 1: A Research Framework for GDSS

as GDSS sessions evolve to geographically-dispersed and asynchronous environments and as the number of session participants increases dramatically.

A possible solution to this problem is an automated monitoring facility which may be able to keep track of the frequency of comment generation. It can then remind individuals to contribute more as needed, or it can suggest ending the current discussion if the frequency of comment generation is decreasing.

2. **Formalized meeting procedures need to be enforced.**

As the number of participants increases in larger group sessions, more rigid control over these sessions may be necessary to maintain order. Even with smaller groups, experience has shown that discussions are often sidetracked onto tangential topics that must be redirected by the facilitator back to the meeting's original problem statement.

The provision of automated parliamentary procedures and automated negotiation techniques may alleviate some of the burden placed upon the group facilitator to direct the discussion. By structuring discussions for more efficiently, automated techniques may improve the productivity and quality of the session.

3. **Facilitators are often uncertain about requirements and options for meeting procedures.**

As new tools and meeting procedures are continually developed for the support of group sessions (and old tools are modified), the requirements for the use of these tools also change. For example, it is often not readily apparent when a particular tool such as Electronic Brainstorming should be used. What size of group does the tool support? What other group, task, and environmental characteristics are assumed to be present for correct use of the tool? In addition to not knowing under what conditions to use a tool, novice facilitators may lack sufficient knowledge of how to use new group support techniques effectively.

Providing an automated counselor to give advice on selecting appropriate group support tools can greatly increase the use GDSSs. By distributing selection expertise, many more groups which do not have expert facilitators will be able to use these systems successfully. Also, novice facilitators and group members can benefit from automated, on-line tutorials on the use of each tool.

4. **Facilitators often have trouble using complex group support systems.**

Even if the facilitator knows when and how to use a particular group support tool, he may encounter problems setting up the tool, linking input to and output from the tool, and correcting any unforeseen "glitches" that may occur during a session. Many specific manual steps are needed to conduct a group session, often with somewhat cryptic, non-intuitive commands or menu items associated with each step. One mistake in the sequence can have disastrous effects later in the group session (such as having to re-start the discussion).

An automated facilitator or facilitator's assistant may be able to manage more of the burden that is currently assumed by the group facilitator to conduct a session. Such an automated facilitator could accept an agenda of group tools selected (see item 3 above) and duration times for each tool as input and could then take the necessary sequence of steps to conduct the group session. In addition, expertise on the correction of unexpected problems could be included, available to the facilitator at the touch of a key.

5. **Group members need access to organizational and environmental information.**

When a group is discussing a highly technical or detailed topic such as budget preparation or project requirements generation, group members may be forced to rely on memory to recall exact financial figures or planning specifications. However, few participants can be anticipated to reliably recall such information.

Since the quality of a group discussion is greatly contingent upon the quality of information brought to the session by group members, the provision of ancillary on-line information retrieval services may be able to augment members' recall of pertinent knowledge. Additionally, intelligent or heuristic search capabilities can be expected to increase the availablity of information internal or external to the organization while decreasing the retrieval time.

Presentation of retrieved information in a format compatible with a group member's mental model is also important. For example, the system should be able to present information to the participant in graphical, textual, or numerical form — whichever is more comfortable to the user.

Research on GDSSs has included the study of methods to make these systems more intuitive and easier to use. However, more needs to be done. As indicated above, one approach to increasing the effectiveness and efficiency of group support is the provision of automated support tools which incorporate human expertise — otherwise known as expert systems.

2.2 The Expert System Solution

Artificial intelligence may be the most important technical contributor to the future of GDSS [14, 37]. Specifically, expert systems (ESs) may transform GDSSs from passive agents that process and present information to active agents that enhance interactions [21]. The goal, however, is to design such integrated systems in a way that enhances desirable procedural and social group processes. This transformation to "softer software" [37] includes the integration of expertise to simplify use and operation of GDSSs.

The expert system paradigm has found many successful commercial applications in a variety of areas [13, 44, 50, 52, 80], and the expert system development methodology continues to be studiously researched as a promising frontier of artificial intelligence. ESs are being generated for new applications at a frantic pace. These systems are being created to fill a great need for decision support and to provide more powerful information systems. Most work to date has been focused on applying ES technology to traditional data processing (DP) areas [27]. Turban and Watkins [71] expanded the scope of ES research by developing a framework for applying ESs to decision support systems (DSSs). The next logical step is to apply ES technology to GDSSs *(Figure 2)*. By integrating these two technologies, GDSSs may become more efficient and effective in a variety of group support environments. However, this integrated system development should be guided less by the available technology and more by the need to understand the fundamental nature of automated group support as well as the group's real behaviors and needs [18].

An additional motivation for integrating ESs with GDSSs is provided by DeSanctis and Gallupe [16]

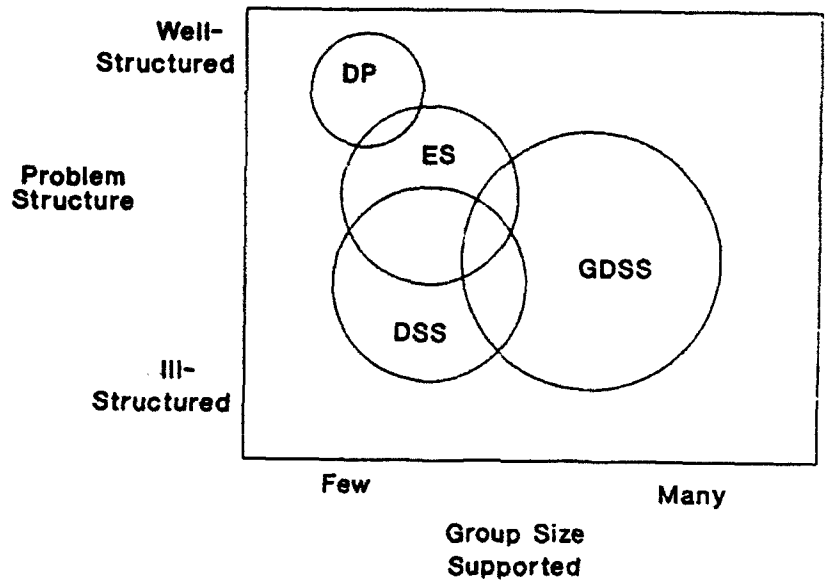


Figure 2: Evolving to ES/GDSS

who have urged that GDSS research proceed along an orderly hierarchy of increasingly technologically-advanced group support systems. DeSanctis and Gallupe have argued that GDSS research should begin with a study of Level 1 systems (systems which provide a communication medium only) and Level 2 systems (systems which provide decision-making support) before proceeding to higher-level systems. They assert that research should advance to Level-3 systems only after the impacts of lower-level systems are thoroughly studied. In this paper, we argue that the time for evolution to Level-3 systems has arrived. Many current GDSS tools provide the communication support of Level 1 systems and the decision-making support of Level 2 systems and have been reviewed thoroughly by the literature [42, 79]. Although more research into these lower-level systems can and should continue, additional gains in knowledge of automated group work can be achieved through the development of higher-level systems. An integration of ESs with GDSSs will provide the knowledge-based support of Level 3 systems which can control the pattern, timing, or content of group information exchange, greatly alleviating the problems of current GDSSs discussed in section 2 above.

Although some initial research has been conducted in the area of applying intelligent support to GDSSs, much more needs to be done. One of the first attempts was made by Stodolsky [66] who developed a prototype system for automated group conflict resolution. Later, Hiltz and Turoff described a system which actively filters and structures information exchange [30]. Similar attempts to add intelligent support to group information exchange were made by Chang and Leung [12], Hogg [31], Malone et al [49], Motiwalla

Attributes	GDSS	ES
Objective	Group decision & communication support	Human expertise replication
Nature of support	Groups	Personal and groups
Who makes the recommendations or decisions?	The group	The system (advisory only)
Characteristics of problem area	Complex, broad, unstructured	Narrow domain, structured
Nature of problems	Ad-hoc, unique	Repetitive
Database/Knowledge base content	Factual knowledge	Factual procedural and knowledge
Reasoning capability	None	Yes (e.g. deduction & induction)
Explanation capability	None or very limited	Yes (how and why capabilities)

Table 1: Differences Between GDSS and ES

Synergistic Area	GDSS Contribution	ES Contribution
Database Management System	May provide distributed database	Improves construction, operation, and maintenance of DB & DBMS
Model Base Management System	Provides standard models	Improves construction, operation, and maintenance of models
Interface	Group interface	Provides explanations, makes friendlier interface
Network Interaction	Geographically distributed group	Improves network performance
Overall Synergy	Provides group support tools	Provides intelligent advice efficiently, expands system capabilities

Table 2: GDSS and ES Synergies

[54], and Pollock [57]. Other efforts have provided only general architectures and research guidelines [2, 32, 36]. Our research builds upon these earlier attempts to provide a more comprehensive view of the problem of automated group support.

2.3 Synergies of ES and GDSS

The great majority of expert systems are stand-alone, independent systems, advising users on specific problem areas. However, one trend in expert systems is toward the development of large, complex systems, far beyond the scope of stand-alone systems [27]. The GENESIS package, for example, includes seven different expert systems which act together to assist in DNA structural analyses. Other examples of complex, integrated systems have been provided by Japan's Fifth Generation project [11].

As was noted earlier, integrating ESs with DSSs has created some powerful and useful information systems [28, 61, 69, 70, 71]. This evolution toward integrating relatively-disparate information technologies has provided the opportunity to capitalize on synergies previously unavailable. Proceeding beyond single-user DSSs to group DSSs will provide even more benefits.

By taking advantage of the heuristic expertise provided by ES technology and the communication and decision making support for groups provided by GDSSs *(Table 1)*, many benefits may be obtained. Integrating ESs with GDSSs provides synergies with respect to database management, model base man-

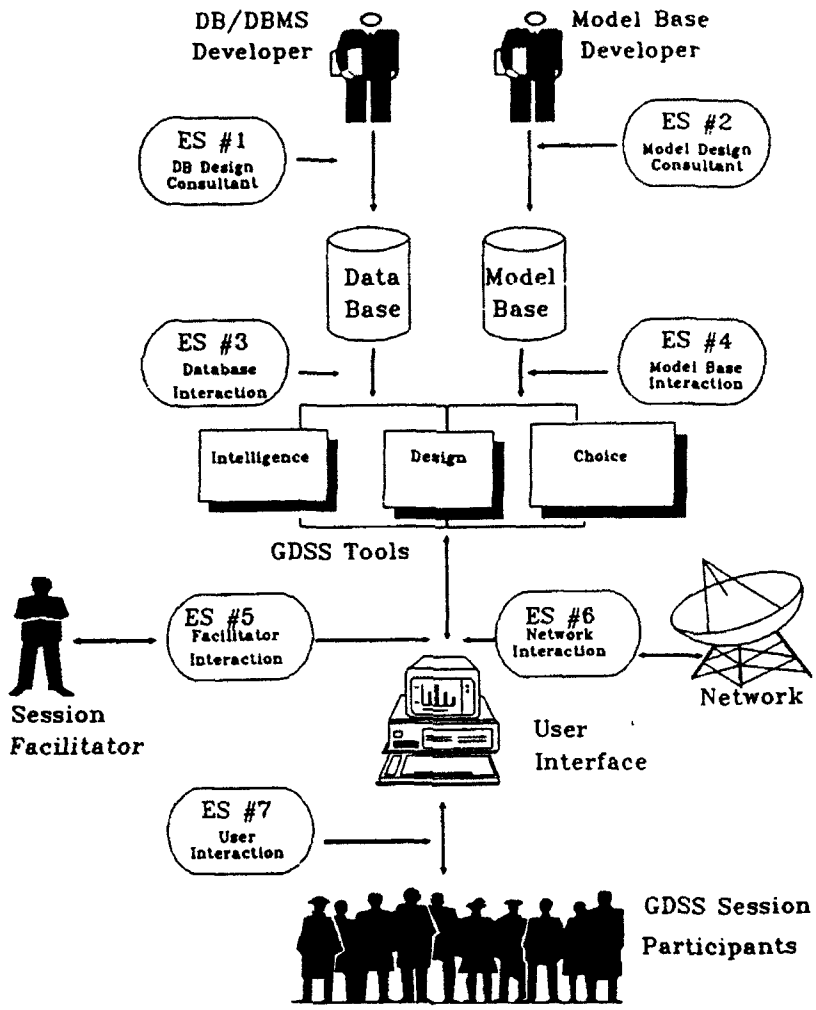


Figure 3: Integration of ES and GDSS

agement, user interface, system development, and system monitoring *(Table 2)*. By incorporating ESs throughout the GDSS architecuture, the overall system is greatly enhanced. Some possible methods for integrating ESs with GDSSs are detailed in the following section.

3 ES Integration Into GDSS Components

A general-purpose GDSS may be described as a system with five principal components: a database, a model base, an uderlying computer network, an interface, a facilitator, and a group of users. Expert systems can be integrated with each of these components as shown in *Figure 3* and described below.

3.1 ES#1: Expert System as a Consultant to the Database Builder

Database linkages with GDSSs are gradually becoming more prevalent as the need for environmental and organizational information scanning increases. During a group session, the ability for participants to

access on-line information can greatly contribute to the quality of the on-going discussion.

The development of a database and a database management system (DBMS) is a formidable task, however, even for experts. Replication of this scarce expertise would be beneficial to the vast number of relatively novice DB/DBMS developers who are faced with this task.

Some important work has already been accomplished with this goal of integration in mind. Jarke and Vassiliou [35] have described how an ES can be used to improve the construction, operation, and maintenance of a DBMS, and Higa [29] and Liu Sheng [47] have detailed automated methods for relational database development. Additional work on the automated creation of databases has been conducted by Berman [10] and Storey [67].

3.2 ES#2: Expert System as a Consultant to the Model Base Builder

Perhaps the most critical component of a GDSS is a model base of tools and procedures sufficiently broad in scope to ensure continuing use of the system [33]. The development of this model base must be approached with some care. As with a database, an ES can greatly assist the model builder in improving the construction, operation, and maintenance of a model base. For a GDSS, a model base may be considered as the set of procedures and software tools that support a group in communication and decision making. However, the steps in the development of these procedures and tools is not well-known. Huber [33] has provided some initial design guidelines, but detailed techniques are lacking. Eventually, GDSS generators (tools from which specific GDSSs for particular group and task environments are constructed) may become available, analagous to the development of DSS generators [34, 63].

GDSSs must be designed for use by different groups and must be flexible to accomodate a variety of group behaviors and tasks. For example, research has shown that two different groups performing the same task use group technology in very different ways [59]. Applegate [6] used manual standard software engineering techniques to design, implement, and evaluate the technical feasibility of automated support for electronic brainstorming. An example of using intelligent techniques for the construction of a GDSS has been provided by Aiken and Hayes [3].

3.3 ES#3: Expert System Interaction with the DBMS

Extracting information, as opposed to facts, from a database can in some cases be a formidable challenge. As the amount of data stored in organizational archives increases exponentially, the ability to access critical information from this resource becomes even more important. The development of sopisticated query languages such as Structured Query Language (SQL) and natural language interfaces such as

HEARSAY, INTELLECT, and CLOUT has alleviated the problem of expressing a searcher's true intent somewhat, but additional difficulties remain. For example, an ES may be able to direct a searcher's path to information which may be relevant but not specifically requested. An ES can provide meeting participants with on-line assistance in retrieving information pertinent to the discussion from the database. The heuristic search capabilities of such a system can provide users with information more efficiently.

In addition to assistance with user queries, an ES may be able to assist the DBMS with the problem of accessing information in a complex, distributed database. Research into the challenges imposed by these networks of data may reveal further areas requiring heuristic expertise.

Johansen [37] posits the concept of artificially-intelligent *agents* which are capable of acting on behalf of humans. These agents may search through a database to find relevant information to a conversation. *Mailer Demons* and *News Agents* are examples of such agents which redirect mail and search out relevant news for a particular user.

Jarke and Vassiliou [35] have described a prototype project at NYU that has coupled an ES with a DBMS of a life insurance firm. PROPHET [53] is another example of a coupled ES/DB which allows users with little or no system knowledge to access information easily and quickly. Finally, the ESP system (described in subsection 4.2 below) has an ES for selecting group members for a potential GDSS session from a database and provides a further example of an ES #3 [4, 5].

3.4 ES#4: Expert System Interaction with the Model Base

Model Management Systems (MMSs) have been researched at length in a variety of contexts, particularly with respect to DSSs [20]. MMSs have been proposed to facilitate management of organizational planning models in a manner similar to the management of organizational data [40].

Choosing the appropriate communication or decision support model for a given group, task, and environment can be a daunting task. Over 70 different group problem solving techniques are available in academic or commercial settings [74], and the nature of the tasks supported vary considerably [43]. Huber notes, however, that the success of a particular GDSS depends on the number and nature of GDSS capabilities, as well as the number and nature of tasks supported [33]. Therefore, some means of optimizing the choice of a particular GDSS technique for a given situation is necessary. Currently, however, little knowledge is available to make these decisions.

Integrating an ES with a GDSS model base will allow even novice group faciliators to select the appropriate tools and procedures for a group work session. This integration will allow the GDSS to provide a sufficient number of tools and adequate support to achieve the critical threshold necessary for

a high frequency of use, which in turn, is necessary for a successful GDSS [33]. ESP is perhaps the only existing system to tackle this problem of selecting the appropriate GDSS technique directly [4, 5].

3.5 ES#5: Expert System Interaction with the Facilitator

In addition to the tasks of selecting the appropriate GDSS tools and group participants for a given session, the facilitator is faced with numerous other administrative and technical tasks. For a general-purpose GDSS, these tasks may include logging-in participants, deleting and creating disk files, clearing the system, etc. Athough these functions may be menu-driven, novice facilitators may need assistance with the sequence of steps involved.

Monitoring of the group discussion is also important for control of the session, and several systems have been proposed to help the facilitator with this task. Process advisors include organizational protocol guides, commitment coordinators, conflict managers, group process coaches, and group processing aides [37] and may have a limited understanding of the subject matter under discussion. Interaction trackers help participants identify patterns in the discussion and may keep track of who has sent a message to whom and how often. Participation ticklers, intelligent comment chainers (chronologically, by subject, or by opinion), and goal achievement aids (task ticklers and motivational advisors) are additional examples of intelligent support for the monitoring task.

Liza [24] is one example of an intelligent agent which monitors session activity and suggests changes of topic or a break. The coordination system described by Ellis et al [21] is an additional example of ways systems may inform users of the states of their actions and may generate automatic reminders and alerts. ESF (described in subsection 4.3 below) is another example of intelligent, automated support for facilitation [48].

3.6 ES#6: Expert System Interaction with the Network

The backbone of a GDSS is a network interconnecting group participants, the facilitator, and supporting hardware in a local or geographically-distributed environment. Many problems arise when trying to communicate over long distances including determining the least-cost communication path as well as establishing and maintaining connections.

Adaptive networks have been devised which are capable of shifting connections in response to changing use patterns [37]. Many systems are already in existence which rely heavily on expert system technology to support network communciation. However, these systems typically act as advisors to human operators rather than operate autonomously.

3.7 ES#7: Expert System Interaction with the User

Expert systems can assist with tool/user interaction in a variety of ways:

- **Natural Language Interface**

  With the advent of natural language interfaces, group participants will feel less inhibited about using a GDSS (many group members may not even know how to type, much less how to use a computer). Many current GDSSs rely on menu bars or special function keys. Facilitators or their assistants must spend a considerable amount of time with new system users just familiarizing them with the tool functions. Intelligent interface support could reduce the time necessary to begin meaningful group work.

  As was noted earlier, several systems have been developed to facilitate user communication with the system via some form of natural language interface. HEARSAY I, for example, uses a limited form of speech understanding for simple database queries and INTELLECT relies on typed entries. On a more restricted level, HAL for Lotus Corporation's Lotus 1-2-3 spreadsheet programs reduces the need for the user to remember specific commands. Much more needs to be done, however, before natural language will be widely available to software systems.

- **Intelligent Mail**

  Electronic mail management systems are becoming increasingly important to GDSSs in which participants are geographically dispersed. Even when participants are in a face-to-face decision room environment, a large group often precludes effective verbal exchange of information. The problem is, however, group members may wish to remain anonymous. Alternatively, many participants may be relatively unknown to the remainder of the group. An intelligent mail system may alleviate the problem of communicating with anonymous group participants. For example, a participant may address his mail *to whom it may concern* rather than by specific addressees. By analyzing keywords in the subject line or body of the message the intelligent mail system may send the message to the appropriate group members. Malone et al [49], Motiwalla [54], and others have developed intelligent mail systems to facilitate message distribution and management, but little has been done to incorporate these systems into a GDSS.

- **Intelligent Help**

  Context-sensitive help would greatly alleviate new group members' difficulties in adjusting to a new GDSS. Intelligent help may be able to infer the user's goal and may be able to direct the user to

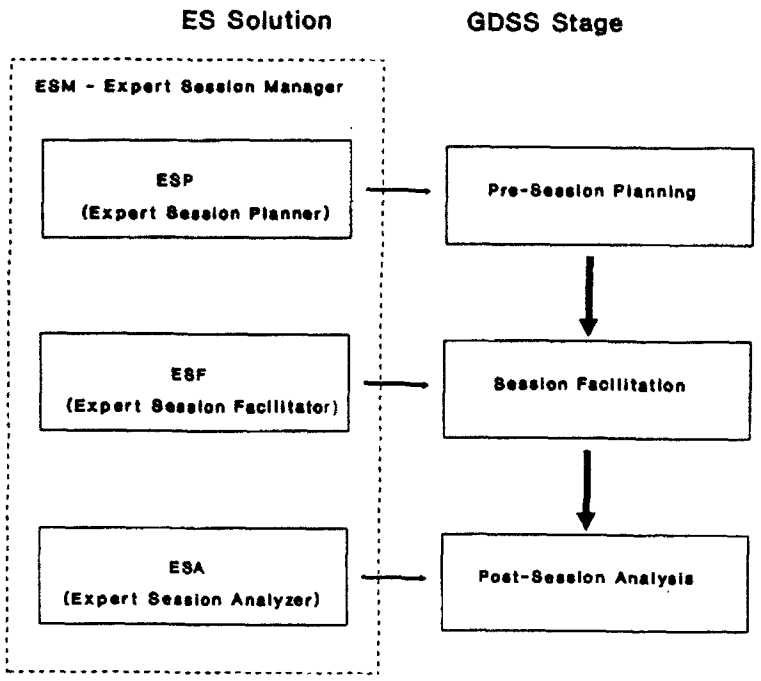


Figure 4: The Stages of a GDSS

related, but not specifically-requested information. A more comprehensive form of intelligent help could take the form of an intelligent tutoring system.

4 A Prototype System for ES/GDSS Integration

Expert Session Manager (ESM) is a prototype system which incorporates three distinct expert systems for supporting each stage of a general-purpose GDSS session *(Figure 4)*. Each of these stages and its expert system support is described in more detail below.

4.1 Pre-Session Planning

During this stage, a session facilitator talks with a representative from the organization which needs to meet to resolve a problem. This representative provides the facilitator with information regarding the group characteristics and the task faced. Using this information, the facilitator can construct an agenda of GDSS tools and meeting times to bring about a successful outcome. In addition, the facilitator can help the representative come up with a list of appropriate group participants from the organization to effectively address all issues of the meeting.

Selecting the correct tools and session participants becomes increasingly important as GDSSs evolve to distributed systems in which a facilitator is not physically present [55]. GDSS support is needed to

resolve the uncertainty involved in making these group and tool selections.

Expert assistance at this stage can alleviate the burden placed upon a human facilitator, enhance the quality and outcome of the ensuing session, and distribute the scarce expertise of GDSS pre-session planning [4, 5]. Effective intelligent support for pre-session planning is becoming even more crucial to computer-based meetings as the trend toward distributed GDSSs grows (due in part to the cost and time advantages over its face-to-face counterpart).

Expert Session Planner (ESP) is a prototype ES designed to support GDSS session facilitators during critical pre-session planning. ESP allows scarce human facilitator knowledge to be distributed and provides consistency across sessions. In addition, ESP provides training for novice session facilitators in the selection of GDSS software.

4.2 Session Facilitation

In this stage, the facilitator follows the agenda prepared in the planning stage and leads the group through the session using appropriate techniques for the task at hand. Some GDSS techniques include: generation of ideas or plans, organization of ideas, consolidation and focusing of ideas, generation of proposals, and assessment of proposals [16]. These techniques can be applied to decision types described as Planning (generation of action-oriented plans), Creativity (generation of novel ideas), Intellective (selection of the correct alternative), Preference (selection of an alternative for which there is no correct answer), Cognitive Conflict (resolution of conflicting viewpoints), and Mixed Motive (resolution of conflicting motives or interests). A variety of GDSS tools are available to support these techniques and decision types including the University of Arizona's GroupSystems GDSS tools: Electronic Brainstorming (EBS), Automated Delphi, Nominal Group Technique (NGT), Issue Analyzer, Issue Organizer, Policy Formation, Vote Selection, Alternative Evaluator, Topic Commenter, Stakeholder Identification and Assumption Surfacing (SIAS), and Enterprise Analyzer.

During the facilitation stage of a GDSS session, the group facilitator must monitor the ongoing discussion to help lead it in appropriate paths of discourse and to terminate the session as comments become redundant or superfluous. As groups grow large, facilitators may be overwhelmed by the sheer volume of information. ES support is needed to ease the burden of facilitation.

In addition to facilitators, group members may need assistance using a GDSS. An organization GDSS may be only infrequently used, resulting in less familiarity with the system and even less use [33]. ES support in the form of on-line tutorials, intelligent help, and other functions may be needed to keep users afresh.

Expert Session Facilitator (ESF) provides some support in this stage by relieving the human facilitator of some of the burden of monitoring comments. This becomes increasingly important when groups and group members are distributed temporally and geographically. ESF monitors the number of comments from each user and may send reminders to contribute more. Additionally, ESF provides an indication to the human facilitator that comments are dropping off in frequency and content, possibly indicating the need to terminate the session.

4.3 Post-Session Analysis

At the end of the session, the facilitator meets with the group representative again to ensure that the session effectively met the group's needs. The post-session analysis includes summary reports of decisions that were made during the meeting as well as a complete transcript of comments generated by the group participants. This output can serve as input to other software such as CASE tools, databases, etc. ES support is needed here to impose structure on this output. Expert Session Analyzer (ESA) assists the human facilitator in structuring this information for more detailed analysis.

4.4 Discussion

In summary, Huber [33] notes that the success of a particular GDSS depends on the range of tasks supported and its frequency of use (both of which are mutually dependent). Through the added functionality and flexibility of a chauffeured, asynchronous, distributed GDSS, ESM provides a sufficient number of tools and adequate support to achieve the critical threshold necessary for a high frequency of use.

Huber also states that effective facilitation depends upon the technical competence of the users of a GDSS and knowledge of the planning process. ESP adds to users' technical competence by providing the most appropriate tool for a given task and group, ESF provides facilitator assistance in monitoring session comments, and ESA provides assistance in analyzing these comments. The most important contribution of ESM, however, is the addition and dissemination of knowledge about the planning, facilitation, and post-session analysis processes. The prototype satisfies this requirement through accumulated knowledge of the requirements, purposes, and goals of the group to be facilitated. The type of organization, its methods and goals, and its purposes for the session are all important for the facilitator (human or automated-chauffeur) to know. Knowledge about the initiator and members participating is also important. The prototype system meets all of these requirements.

5 Conclusion

A review of the literature as well as extensive experience conducting group support sessions has indicated several problems with GDSS user interfaces, group communication, and access to environmental information — all requiring human expertise to correct. This paper has provided a general framework in which to address these problems by integrating expert systems with group decision support systems, combining the advantages of each and achieving a greater synergy overall. This framework can be used for the study of current and development of future automated group support systems. A highly-integrated ES/GDSS prototype system under development at the University of Arizona was detailed as an example approach for such future systems.

In summary, an ES can make a GDSS a more flexible and powerful aid for group support. Existing GDSSs incorporating a wide variety of tools are limited by their ease of use and by the scarce expertise of human facilitators; integrating ESs with GDSSs will greatly simplify the use of such systems and will enable the replication of GDSS session management knowledge for remote, distributed session planning.

Scott Morton has noted that conventional DSSs may be supplanted by EDSSs (expert decision support systems) [61]; the same may be said for GDSSs. ESs may some day monitor a group's decision processes, analyze the content of the discussion, and direct the group on alternate paths to reach a desired goal.

Humans will never completely disappear, however, as there will continue to be a need for "hand-holding," the reassuring presence of a human expert to extract group participants from difficult quagmires and imbroglios. Also, for small groups with a limited need for a vast array of GDSS tools, a human facilitator may still be the most cost-effective means of controlling group collaborative work.

Potential developers of an integrated ES/GDSS must keep in mind the many limitations of the respective technologies to assure a successful outcome. Particularly, additional research is required in the area of deep knowledge, the underlying theory of group processes, to adequately represent the human behavior emulated by an ES group facilitator. The ESM prototype as well as the framework present a panoply of rich research areas including technical, behavioral, and design issues which are as yet relatively unexplored. Work is currently under way to provide intelligent support for the facilitation and post-session analysis stages of GDSS sessions through natural language processing and additional ES features. In addition, more rules are being added to the tool selection module of ESP to incorporate knowledge about the nascent field of asynchronous, distributed GDSS.

REFERENCES

1. *Plex Guide*, Department of Management Information Systems, University of Arizona, 1988.

2. AGARWAL, R. AND PRASAD, K. Enhancing the group decision making process: An intelligent systems architecture. In *Proceedings of the Twenty-Second Annual Hawaii Conference on System Sciences* (Kailua-Kona, Hawaii, January 3-6), *1989*, 3, 273-279.

3. AIKEN, M. AND HAYES, G., A DEVS-Scheme simulation of an electronic meeting system. *Simulation Digest* 20, 2 (Summer 1989), 31-39.

4. AIKEN, M., MOTIWALLA, L., LIU SHENG, O., AND NUNAMAKER, J. An expert systems approach to group decision support systems planning. In *Proceedings of the 1989 Annual National Conference of the Association of Computer Educators* (Denver, Colorado, September), 1989, 96-104.

5. AIKEN, M., MOTIWALLA, L., LIU SHENG, O., AND NUNAMAKER, J. ESP: An expert system for pre-session group decision support systems planning. In *Proceedings of the Twenty-Third Hawaii International Conference on System Sciences*, (Kailua-Kona, Hawaii, January 2-5), 1990, 3, 279-286.

6. APPLEGATE, L. *Idea management in organization planning.* Unpublished PdD dissertation, University of Arizona, 1986.

7. APPLEGATE, L. M., KONSYNSKI, B. R., AND NUNAMAKER, J. F. A group decision support system for idea generation and issue analysis in organizational planning. In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Austin, TX, December), 1986, 16-34.

8. BASKIN, A. B., LU, S., STEPP, R. E., AND KLEIN, M. Integrated design as a cooperative problem solving activity. Working paper, University of Illinois, 1988.

9. BEAUCLAIR, R. A. An experimental study of the effects of GDSS process support applications on small group decision making. Unpublished Ph.D. dissertation, Indiana University, 1987.

10. BERMAN, S. A semantic data model as the basis for an automated database design tool. *Information Systems* 11, 2 (1986), 149-165.

11. BONCZEK, R. H., HOLSAPPLE, C., AND WHINSTON, A. *Developments in DSS*, Research Report 1ST-8108519, MIS Research Center, Krannert Graduate School of Management, Purdue University, 1984.

12. CHANG, S. AND LEUNG, L. A knowledge-based message management system, *ACM Transactions on Office Information Systems* 5, 3 (July 1987), 213-236.

13. CLANCEY, B. C. AND SHORTLIFFE, E. H., eds. *Readings in Medical Artificial Intelligence: The First Decade*, Addison-Wesley, Reading, Massachussetts, 1984.

14. CROWSTON, K. AND MALONE, T. Intelligent software agents. *Byte* 13, 13 (December 1988), 267-272.

15. DENNIS, A., GEORGE, J., JESSUP, L., NUNAMAKER, J. AND VOGEL, D. Information technology to support electronic meetings. *MIS Quarterly* 12, 4 (December 1988), 591-624.

16. DESANCTIS, G. AND GALLUPE, B. A foundation for the study of group decision support systems. *Management Science* 33, 5 (May 1987), 589-609.

17. DESANCTIS, G. AND GALLUPE, B. Group decision support systems: A new fronteir. *Data Base* (Winter 1985), 3-10.

18. DOYLE, J. Expert systems without computers or theory and trust in artificial intelligence. *The AI Magazine* 5, 2 (Summer 1984), 59-63.

19. EASTON, A. An experimental study of automated versus manual support for stakeholder identification and assumption surfacing with small groups. Unpublished Ph. D. dissertation, University of Arizona, 1988.

20. ELAM, J. AND KONSYNSKI, B. Using artificial intelligence techniques to enhance the capabilities of model management systems. *Decision Sciences* 18, 3, 487-502.

21. ELLIS, C., GIBBS, S., AND REIN, G. Groupware: The research and development issues. *Report STP-414-88*, MCC Software Technology Program, Austin, Texas, December 1988.

22. FEIGENBAUM, E. A., BUCHANAN, B. G., AND LEDERBERG, J. On generality and problem solving: A case study using the DENDRAL program. *Machine Intelligence* 6, (1971) B. Meltzer and D. Michie, eds., American Elsevier, New York, 165-190.

23. GALLUPE, B. The impact of task difficulty on the use of a group decision support system. Unpublished Ph. D. dissertation, University of Minnesota, 1985.

24. GIBBS, S. J. LIZA: An extensible groupware toolkit. *Proceedings of the CHI '89 Conference on Human Factors in Computing Systems*, (Austin, TX, April 30 - May 4), ACM, New York, 1989.

25. GRAY, P., BERRY, N. W., ARONOFSKY, J. S., HELMER, O., KANE, G. R., AND PERKINS, T. E. The SMU decision room project, In *Transactions of the First International Conference on Decision Support Systems* (Atlanta, Georgia, June), 1981, 122-129.

26. HALE AND HASEMAN. EECS: A prototype distributed executive communication and support system. In *Proceedings of the Twentieth Hawaii International Conference on System Sciences*, (Kailua-Kona, Hawaii, January), 1987, 3, 557-565.

27. HARMON, P. AND KING, D. *Expert Systems: Artificial Intelligence in Business*, John Wiley & Sons, New York, New York, 1985.

28. HENDERSON, J. C. Finding synergy between decision support systems and expert systems research. *Decision Sciences* 18, 3 (Summer 1987), 33-349.

29. HIGA, K. AND LIU SHENG, O. R. Intelligent database development: An AIMAIL example. Technical Report, Department of Management Information Systems, University of Arizona, 1987.

30. HILTZ, S. R. AND TUROFF, M. Structuring computer-mediated communication systems to avoid information overload. *Communications of the ACM*, 28, 7, (1985), 680-689.

31. HOGG, J. Intelligent message systems. In *Office Automation*, D. TSICHRITZIS, Ed., Springer-Verlag, New York, 1985, 113-133.

32. HUBER, G. P. Group decision support systems as aids in the use of structured group management techniques. In *DDS-82 Conference Proceedings*, (1982), 96-108.

33. HUBER, G. P. Issues in the design of group decision support systems. *MIS Quarterly* (September 1984), 195-204.

34. HWANG, S. Automatic model building systems: A survey. In *DSS '85 Transactions*, Providence, RI, Institute of Management Sciences, 1985.

35. JARKE, M. AND VASSILIOU, Y. Coupling expert systems with database management systems. In *Artificial Intelligence Applications for Business*, W. REITMAN, ed., APLEX Publishing Corporation, Norwood, New Jersey, 1984.

36. JARKE, M. Knowledge sharing and negotiation support in multiperson group decision support systems. *Decision Support Systems* 2 (1986), 93-102.

37. JOHANSEN, R. *Groupware: Computer Support for Business Teams*, Free Press, New York, 1988.

38. KEEN, P. AND SCOTT MORTON, M. *Decision Support Systems: An Organizational Perspective*, Addison-Wesley, Reading, MA, 1978.

39. KERR, E. AND HILTZ, S. *Computer-Mediated Communication Systems*, Academic Press, New York, 1982.

40. KONSYNSKI, B. AND DOLK, D. Knowledge abstractions in model management, *DSS-82 Transactions* (1982).

41. KOTTEMAN, J. AND KONSYNSKI, B. Information systems planning and development: Strategic postures and methodologies, *Journal of Management Information Systems* 8, 3 (1983), 195.

42. KRAEMER, K. L. AND KING, J. L. Computer-based systems for cooperative work and group decision making. *ACM Computing Surveys* 20, 2 (June 1988), 115-146.

43. KRAEMER, K. L. AND KING, J. L. Computer-based systems for cooperative work and group decision making: Status of use and problems in development. In *Proceedings of the Conference on Computer Supported and Collaborative Work* (October 1986), 353-375.

44. KULIKOWSKI, C. A. Artificial intelligence methods and systems for medical consultation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (September 1980).

45. KULL, D. Group decisions: Can a computer help? *Computer Decisions* 14, 5 (May 1982), 70-84.

46. LEWIS, F. Facilitator: A microcomputer decision support system for small groups. Unpublished Ph. D. dissertation, University of Louisville, 1982.

47. LIU SHENG, O. R. A structured approach for relational database development. Technical Report, Department of Management Information Systems, University of Arizona, 1987.

48. LIU SHENG, O., AMARAVADI, C., AIKEN, M., AND NUNAMAKER, J. IOIS: A knowledge-based approach to an integrated office information system. Working paper, University of Arizona, 1989.

49. MALONE, T. W., GRANT, K. R., TURBAK, F. A., BROBST, S. A., COHEN, M. D. Intelligent information systems. *Communications of the ACM* (May 1987), 390-402.

50. MCDERMOTT, J. R1: A rule-based configurer of computer systems. *Artificial Intelligence* 19, (September 1982), 39-88.

51. MCINTYRE, S., KONSYNSKI, B., AND NUNAMAKER, J. Automated planning environments: Knowledge integration and model scripting. *Journal of Management Information Systems*, Forthcoming.

52. MICHAELSON, R. AND MICHIE, D. Expert systems in business. *Datamation* 29, 11 (November 1983), 240-246.

53. MORRISON, J. AND MORRISON, M. PROPHET: A flexible, maintainable and expandable object-oriented knowledge base/database system. Working paper, University of Arizona, 1990.

54. MOTIWALLA, L. A knowledge-based messaging system: Framework, design, prototype development, and validation. Unpublished Ph. D. dissertation, University of Arizona, 1989.

55. NAGAO, D., PARSONS, C., HEROLD, D. Group dynamics in distributed computer supported groups. Working paper, College of Management, Georgia Institute of Technology, 1989.

56. NUNAMAKER, J., VOGEL, D., HEMINGER, A., MARTZ, B., GROHOWSKI, R., AND MCGOFF, C. Experiences at IBM with group support systems: A field study. *Internation Journal of Decision Support Systems* 5, 2 (June 1989), 183-196.

57. POLLOCK, S. A rule-based message filtering system. *ACM Transactions on Office Information Systems* 6. 3 (July 1988), 232-254.

58. QUINN, R. E., ROHRBAUGH, J., AND MCGRATH, M. R. Automated decision conferencing. How it works. *Personnel* 62, 11 (November 1985), 49-55.

59. REIN, G. AND ELLIS, C. The Nick summer experiment: A field study on the usage of meeting support technology by software design teams. *Report STP-018-88*, MCC Software Technology Program, Austin, Texas, 1988.

60. RICHMAN, L. Software catches the team spirit. *Fortune* 115, 12 (June 8, 1987), 125-136.

61. SCOTT MORTON, M. Expert decision support systems. a paper presented at the special DSS conference, Planning Executive Institute and Information Technology Institute, New York, New York, May 21-22, 1984, 21.

62. SIMON, H. A. *The New Science of Management Decision*, New York: Haper and Row, 1960.

63. SPRAGUE, R. H., Jr. A framework for research on decision support systems. *MIS Quarterly* 4, 4 (December 1980), 1-26.

64. STEEB, R. AND JOHNSTON, S. C. A Computer-based interactive system for group decision making. *IEEE Transactions on Systems, Man, and Cybernetics* SMC-11, 8 (August 1981), 544-552.

65. STEFIK. The knowledge medium. *The AI Magazine* 3, 1 (Spring 1986).

66. STODOLSKY, D. Automating mediation in group problem solving. *Behavior Research Methods and Instrumentation* 13, 2 (1981), 235-242.

67. STOREY, V. C. An expert view creation system for database design. Unpublished PhD dissertation, University of British Columbia, Vancouver, B.C., Canada, October 1986.

68. STRAUB, D. AND BEAUCLAIR, R. Current and future uses of group decision support system technology: Report on a recent empirical study. *Journal of Management Information Systems* 5, 1, (Summer 1988), 101-116.

69. SULLIVAN, G. AND FORDYCE, K. Decision simulation, one outcome of combining AI and DSS. Working paper #42-395, IBM Corporation, Poughkeepsie, New York, 1984.

70. SULLIVAN, G. AND FORDYCE, K. The Role of artificial intelligence in decision support systems. A paper delivered at the International meeting of TIMS in Copenhagen, Denmark, June 17-21, 1984.

71. TURBAN, E. AND WATKINS, P. Integrating expert systems with decision support systems. *MIS Quarterly* 10, 2 (June 1986), 121-134.

72. TUROFF, M., FOSTER, J., HILTZ, S., NG, K., WALSH, K., AND JOHAR, G. The TEIES design and objectives: Computer mediated communications and tailorability. Working paper, Computerized Conferencing and Communications Center, New Jersey Institute of Technology, Newark, NJ, 1988.

73. TUROFF, M. AND HILTZ, S. R. Computer support for group versus individual decisions. *IEEE Transactions on Communications* 30, 1 (January 1982), 82-90.

74. VAN GUNDY, A. B. *Techniques of Structured Problem-Solving*, Van Nostrand Reinhold, New York, 1981.

75. VOGEL, D., NUNAMAKER, J., AND KONSYNSKI, B. Group, task, and technology in a group support system environment. *DSS Journal* Forthcoming.

76. VOGEL, D., NUNAMAKER, J. Group decision support system impact: Multi-methodological exploration. *Information and Management* Forthcoming.

77. WAGNER, G. AND NAGASUNDARAM, M. Meeting process augmentation: The real substance of GDSS. *IFIPS Working Group 8.3 Conference on Organizational Decision Support Systems* (June 1988).

78. WATSON, R. A study of group decision support system use in three and four-person groups for a preference allocation decision. Unpublished Ph. D. dissertation, University of Minnesota, 1987.

79. WATSON, R. T., DESANCTIS, G., AND POOLE, M. S. Using a GDSS to facilitate group consensus: Some intended and unintended consequences. In *Proceedings of 8th ICIS* (1987), 339-402.

80. WINSTON, P. H. AND PRENDERGAST, K. A., eds. *The AI Business: The Commercial Uses of AI*, MIT Press, Cambridge, MA, 1984.

81. ZIGURS, I. The effect of computer-based support on influence attempts and patterns in small group decision-making. Unpublished Ph. D. dissertation, University of Minnesota, 1987.

Appendix III

Application of Knowledge-Based System Design to the Design of a Distributed Group Decision Support System[1]

Joline Morrison
Department of Management Information Systems
University of Arizona
Tucson, Arizona 85721

[1] *This paper was supported in part by a grant from the Army Institute of Research in Management Information, Communications, and Computer Science (AIRMICS), Atlanta, Georgia, Grant #: DAKF-11-88-C-0021.*

TABLE OF CONTENTS

1. Introduction: The Software Problem

As software systems become increasingly complex, the problems associated with software project development increase also. Projects are haunted by schedule and budget overruns, unmet specifications, and inadequate system performance. Many delivered systems are unreliable and unmaintainable. These problems can be caused by developers having a vested interest in tools and techniques that are outdated, ad hoc, ineffective, or inefficient. Developers may also be victims poor project team management [DeMarco and Lister, 1987], and may resist change due to the time pressure of increasing software demand over supply, An additional source of difficulty, the focus of this paper, is the absence of a methodical and disciplined approach to software engineering [Ratcliff, 1987].

Why, after over twenty years of building software, has no methodical and disciplined approach emerged? Brooks [1987] attributes this to four problems inherent to software:

Complexity. Software is highly complex because no two parts are alike (beyond the level of individual statements or subroutines). It is therefore far more complex than computers or automobiles, which contain repeated elements.

Conformity. Much of the complexity associated with software evolves from its requirement to conform to its intended environment in terms of interfaces and existing hardware.

Changeability. In manufactured goods, design changes are usually incorporated into new model releases. However, since software is "infinitely malleable", faster and more numerous modifications are often demanded, and the product is constantly in a state of change.

Invisibility. Abstract models may be used to show system data flows, control flows, timing sequences, and patterns of dependencies. However, any one of these may be difficult to represent in a complex system. Attempting to show the complex inter-relationships of all of them working together requires such a high level of abstraction that the overall system becomes unvisualizable.

These ideas indicate that software systems are highly specialized. This may partially explain why no satisfactory engineering methodology exists: there would need to be as many methodologies as systems. Perhaps a different viewpoint is needed. Most software process models are either *document-driven* or *code-driven* [Boehm, 1988]. Unfortunately, requirements documents and code for all systems are different. However, the objectives are the same: to build a software system to meet a specified set of requirements. Therefore,

it is proposed to apply an objectives-driven engineering approach to the software process to investigate its contribution to a software project.

Descriptions of common software processes and requirements and design specification methods are presented to derive conclusions about their strengths, weaknesses, and domains. An attempt is made to overcome some of these problems by applying an engineering design methodology called **Knowledge-Based System Design** to the software process. This methodology is described, and an adaptation suitable for software design is developed. The resulting process is then illustrated by applying it to a software project that involves building a software system to support group decision-making in a distributed environment. Advantages and disadvantages of using the methodology within this project are presented along with possible future research directions.

2. Overview of Software Process Models

Software process models define tasks within software development projects. Four common approaches (exploratory programming, the waterfall model, formal transformations, and the spiral model [Boehm, 1988; Sommerville, 1989]) and their domains will be described.

2.1 Exploratory Programming

A holdover from the early days of software development, this technique involves writing and modifying code until it works correctly. It is useful for exploratory systems where requirements are difficult to define. A disadvantage is that after a number of fixes, the code is so poorly structured that the bulk of programmer man-hours is spent fixing flaws introduced by earlier fixes [Brooks, 1982]. This approach may be acceptable for small systems with stable requirements; however, systems built using this process are probably best viewed as prototypes for establishing system requirements.

2.2 The Waterfall Model

This model was initially suggested by Royce in 1970 [from Sommerville, p. 7]. Numerous refinements and variations have been suggested, but the basic stages are as follows:

(1) *Requirements analysis and specification.* The prospective system's specifications, constraints and goals are established through interviews with system users. They must then be documented so as to be understandable to a wide range of people, including programmers, system engineers, managers, and users [Trapnell, 1969].

(2) *System and software design.* System design involves partitioning requirements into hardware and software components; software design involves partitioning the software functions into modules that can be transformed into computer programs. Usually elaborate documentation must be generated and approved before implementation may begin.

(3) *Implementation and unit testing.* The software design is implemented into a series of computer programs that are individually verified.

(4) *System testing.* The components are assembled and tested together.

(5) *Operation and maintenance.* The system is put into operation, and improved and enhanced as necessary.

The waterfall model recognizes feedback/alteration loops between stages, with looping allowed only between successive stages in order to minimize re-working.

This approach is well known, widely used, and has well-established standards. However, its requirement that design specifications be frozen beyond the implementation phase can result in software that does not have the desired functionality. Also, preparation of the elaborate requirements and design documentation can be time-consuming and costly. The model is well-suited to large multi-component systems with stable requirements, but does not work well for systems with ill-defined or evolving requirements. The high degree of formality and volume of documentation is desirable for systems that are built and maintained by large programming teams with high personnel turnover.

2.3 Formal Transformations

This model involves creating a formal specification (consisting of sets, mappings, constraints, etc.) of the desired system and then transforming it into executable code [Barstow, 1987]. The transformation preserves correctness, so the resulting program accurately represents its specifications. Since modifications are made to specifications rather than code, the poorly-structured and over-modified code often found in exploratory programming methods is avoided. The design, implementation and testing phases of the waterfall model are condensed into the single phase of transformation.

Presently this is primarily an experimental technique. It is well-suited for systems with evolving requirements, provided that changes do not require extensive re-working of system specifications. However, complex or interactive systems with many potential combinations and/or variations of inputs, outputs, and states may be difficult to specify.

2.4 The Spiral Model [Boehm, 1988]

This model views the development process as a series of concentric spirals, with each cycle representing a progression in the completion of the system. The project path is determined by the amount of perceived risk in areas such as interface design, system performance, project budget, and scheduling.

Each cycle begins with identification of the cycle's objectives, alternatives, and constraints. The alternatives are evaluated in relation to the objectives and constraints in order to identify areas of uncertainty that are potential sources of project risk. The first two cycles, conceptual development and software design, employ resolution strategies such as prototyping, simulating, or administering user questionnaires to resolve risk.

Remaining sources of risk set the path for future progress. For example, if a prototype design has acceptable performance and user-interface risk considerations but unacceptable budget and scheduling risks, the model may proceed along steps similar to the waterfall model, repeating the budget and scheduling evaluation and risk-resolution in the design and implementation phases. Conversely, if the prototype's requirements-change risk is very low, the exploratory method may be used while continuing to evaluate performance and user-interface risk.

The spiral model encourages development in the manner best suited to the particular project. However, it lacks a defined methodology and specifications for milestones and project reviews. It is not well-suited for contract software projects with rigid deliverable deadlines, since development paths are difficult to predict. Until this method is defined more clearly and completely, it is probably best suited to in-house systems.

This overview leads to the observation that the types of systems that can be built using the exploratory, waterfall, or formal transformation processes tend to be limited; and, no process is satisfactory for a wide range of systems[1]. Also, these approaches view software systems from a narrow, requirements-driven perspective rather than from a global, objectives-driven viewpoint with an eye on evolving constraints and future enhancements.

The underlying methods of requirements and design specification within these software processes will now be investigated.

3. Specifying System Requirements

All software processes require determination of the services the system should provide and the constraints under which it will operate. Parnas and Clements [1986] argue that complete requirements documentation is essential before project design begins in order to allow users to review system functionality and behavior, prevent requirements decisions from being made accidentally during the design phase, avoid duplication and inconsistency, make good manpower and time estimates, and ensure project continuity in case of personnel turnover. The form of this documentation can range from a mental model to an elaborate portfolio of text specifications and graphical models. The exploratory, formal transformation, and spiral model processes specify no explicit form for requirements or design documentation, while the waterfall model requires specific documents as process milestones. The process of requirements definition/specification/modelling will be explored

[1] The Spiral Model is more versatile in its range of system types and uses an objectives-driven approach, but it lacks a well-defined methodology.

from the waterfall model's rigorous viewpoint, with the understanding that the other processes require similar but perhaps less formal documents.

3.1 Contents of the Requirements Definition and Specification Documents

Requirements documentation can be produced at different levels of abstraction depending on its purpose and intended audience. Often, two separate documents are created: the *requirements definition* is highly conceptual, written in a narrative style, and aimed towards users. The *requirements specification* is more detailed and technical, often presented as an abstract model, and aimed toward system developers [Sommerville, 1989]. The contents of these documents is described.

3.1.1 Requirements Definition

The degree of detail contained in the requirements definition will vary depending on the organization, project, and process. Broadly speaking, it should state why a system is needed, what functions the system will accomplish, and how the system will accomplish these functions in terms of its functional architecture and design constraints [Ross and Schoman, 1977]. Herninger [1980] maintains that the requirements definition should contain external system behavior specifications, implementation constraint descriptions, and acceptable responses to undesired events such as hardware failures and user errors. She further states that the document should be easy to change, be designed to serve as a reference tool, and record forethought about the system life-cycle (i.e. anticipate future changes).

3.1.2 Requirements Specification

Two types of requirements are defined within the requirements specification document: *functional requirements*, concerning the functionality the system must possess and the nature of its interaction with its environment; and *non-functional requirements*, regarding restrictions on possible solutions [Roman, 1985]. Functional requirements involve specifying the relevant internal states of both the system and its environment. From the user's viewpoint, they can be described as system input and output [Ratcliff, 1988]. Functional

specifications should emphasize the "what" rather than the "how" of the software system [Borgida et al, 1985].

Non-functional requirements are difficult to specify, since they are often not entirely known at the time of initial requirements specification. However, Roman [1985] suggests the following categories:

Interface - how information is presented to and received from the environment;

Performance - performance issues such as response time, throughput, reliability, and security;

Operating - physical attributes and components distribution;

Life-cycle - design quality and development/maintenance/enhancement issues;

Economic - short and long-term cost considerations;

Political - policy and legal issues

Parnas and Clements (1986) suggest *undesired event handling* (management of erroneous or unpredicted events) as an additional non-functional constraint.

The requirements specification document may be produced by adding detail to the requirements definition. However, large and complex systems may require creation of a model to emphasize relevant information and hide insignificant details. System requirements modelling techniques are now examined.

3.2 Modelling System Requirements Within the Requirements Specification

Software systems are composed of two basic components: data and processes. Architectural abstractions used in software modelling include *networks, hierarchies*, and *objects*. *Networks* depict processes as nodes and data as directed arcs between nodes. Processes are assumed to be asynchronously concurrent, so it is difficult to represent real-time systems with this method alone. *Hierarchies* are useful for analyzing system composition. A possible software-system high-level decomposition could be as follows [Ratcliff, 1987]:

Decomposition Level	Component	
	Data	Process
System	Database Subschemas Tables	System Subsystems Programs
Program	File Records Fields	Program Module Instructions

Hierarchical modelling is the basis for *top-down design* (where the problem is first considered at its highest level of abstraction and then successively decomposed into modules) and *bottom-up design* (where low-level modules are synthesized).

Objects represent system components as self-contained environments with local processes and well-defined interfaces that accept and deliver data to one another. Object-oriented modelling is usually used in conjunction with other techniques as a low-level design abstraction since objects are readily translated into program modules that communicate by passing parameters.

Many methods have been developed for modelling and documenting system requirements. Descriptions of a sampling of such methods, identifications of their modelling techniques, analysis of their strengths and weaknesses, and conclusions about their common problems will be presented.

3.3 Methods for Modelling System Requirements

Structured Systems Analysis (SSA). The SSA approach [DeMarco, 1978; Gane & Sarson, 1979] requires the current physical system to be modelled using *data-flow diagrams* (DFDs)[2]. This physical DFD is then converted to a logically-equivalent DFD and modified as

[2] A DFD is a network model that represents processes as nodes, data flows as labelled arcs, and data sources and sinks at the boundaries of the system or subsystem.

necessary to capture the requirements of the new system. Alternative implementations can be generated and evaluated. **Modelling technique:** network. **Strengths:** is a well-accepted methodology, with easily constructed and interpreted diagrams. **Weaknesses:** is difficult to use for programming tasks without well-defined requirements; DFDs provide no information about data types (pure versus control data [Couger et al, 1982]); supplementary tools such as decision trees and decision tables are required to express decision logic; requirements for real-time systems cannot be adequately expressed.

Jackson System Development (JSD). The JSD method [Cameron, 1986] consists of three phases: modelling, networking, and implementation. *Modelling* requires identification of the system's real-world entities and their functional requirements. Actions that affect each entity are entered sequentially below the entity in a hierarchical structure. Actions can lead to new entities, thus expanding the hierarchy. Leaf actions are decomposed into disconnected sequential processes. *Networking* involves expanding the existing processes and creating new processes until the level of detail is to that of a single instruction. A network is then created to show processes and their interconnections. *Implementation* involves specifying process schedules and timing constraints and transforming the specification into an implementation. **Modelling techniques:** hierarchy, network. **Strengths:** emphasizes system environment considerations; is well-suited to real-time systems; provides highly-detailed requirements specification that is easily transformed into design specification. **Weaknesses:** identification of initial entities may be difficult; diagrams are complex and may be difficult to understand; the methodology is complicated and not widely used.

SADT(TM) (Structured Analysis and Design Technique). SADT is a proprietary system that provides a detailed system requirements specification as well as project management features for team coordination, documentation, and project control. Requirements are expressed through hierarchies of networks called "models" that are constructed from a special symbol library, and may include supplementary text. Models can be data oriented or process oriented; they may express either an individual or collective view of the system [Ross, 1977; Ross and Schoman, 1977]. **Modelling technique:** hierarchy of networks. **Strengths:** provides detailed and precise specifications that may be carried to any desired level of decomposition; may be used for most types of systems. **Weaknesses:** is a proprietary and complex methodology that is difficult to learn and understand; the extensive graphics symbol library may not be understood by non-specialists.

SREM (Software Requirements Engineering Methodology). SREM is part of the SYSREM system used to develop time-critical, distributed real-time systems for the Department of Defense. Based on a finite state machine model[3], requirements are represented by a specialized graphical notation that maps processes from the arrival of a given input through

[3] A finite state machine is a network model that represents functions as nodes, data as inputs and outputs of nodes, and control as the mechanism that activates functions in the desired sequence [from Sommerville, p. 75].

to the eventual output. This may be repeated at various levels of abstraction. SREM is supported by Requirements Statement Language (RSL), which takes inputs of syntactically-correct requirements statements and stores them in a database that can be queried to check completeness and consistency [Ratcliff, 1987; Couger et al, 1982]. **Modelling technique:** network. **Strengths:** provides complete and correct specifications that are easily translated into a design specification. **Weaknesses:** developed for a specialized system type; is a complex, proprietary methodology that may require extensive training; database system requires a large amount of computing power.

PSL/PSA (Problem Statement Language/Problem Statement Analyzer). PSL/PSA is a programming language designed to express software requirements [Teichrow and Hershey, 1977]. System requirements are represented in PSL, and then translated and stored in a database that can be queried using one of several PSA report generators. PSL models systems in database-type terms of "objects" that possess "attributes"; objects are related by various "relationships". Objects in PSL can be events and processes as well as data. **Modelling technique:** objects. **Strengths:** generates well-defined machine-readable requirements with high degree of consistency and correctness. **Weaknesses:** PSL is proprietary, highly formal, and structured, with a fairly steep learning curve.

Programming Languages. A problem common to specialized notations such as SADT, SREM, and PSL/PSA is high startup and training cost. As a result, the use of derivatives of high-level programming languages such as Ada(TM) has been suggested. Mander [from Sommerville, p. 95] compared writing requirements definitions in Ada to writing definitions in SADT and PSL/PSA, and concluded that Ada was as expressive as the other notations provided extensive comment statements were included. Ada dialects have been developed specifically for requirements specification [from Ratcliff, p. 42]. **Modelling technique:** networks, objects. **Strengths:** provide structure and formality to specifications. **Weaknesses:** may not be able to express high enough levels of abstraction or avoid implementation details.

The following conclusions can be drawn from this review of requirements specifications methods:

1. Less-complex methods such as SSA and programming languages such as Ada are often not versatile enough to represent types of systems different than those for which they were developed (e.g. transactions processing and real-time embedded control).
2. Complex methods such as SADT or PSL/PSA are difficult to learn and implement, and users may feel that the effort spent writing structured specifications in such languages would be better spent writing executable code.

4. Software Design

Software design is generally accomplished by converting the requirements specification into a detailed design specification using either a *functional* (top-down) or *object-oriented* (combination top-down/bottom-up) approach. These approaches will be examined and compared, and the formal transformation design process will also be considered.

4.1 Functional Design

This method starts with a high-level solution that is gradually refined into a detailed design. Example methods include *structured design* and *step-wise refinement.*

Structured design [Constantine and Yourdon, 1979] is an extension to Structured Systems Analysis that takes a system described by data flow diagrams and implements it as a set of sequential processes. DFD processes are translated into hierarchical graphs called **structure charts** that show the organization of the software in terms of software components and subcomponents; components are represented as rectangles, with labelled inputs and outputs; data stores are represented as rounded boxes, and user inputs are shown as circles. A **data dictionary** contains definitions and descriptions of all system data and processes.

Stepwise refinement [Wirth, 1971] involves expressing system requirements as a series of tasks in English-like syntax. Tasks are successively decomposed and refined first into pseudocode and then into the syntax of an underlying computer language. As tasks are broken down, data is also refined, decomposed, and structured. Design decisions should be decomposed as much as possible to recognize and reduce interdependencies between tasks. Decisions dealing with details of representation should be postponed as long as possible. Alternatives should be weighed carefully, and earlier decisions should be revoked as necessary.

Functional design approaches the design process in a methodical and logical manner. The problem is considered in its entirety but is broken down into manageable tasks. Functional design requires critical decisions to be made early in the development process, however, and errors can result in costly fixes or even re-working. Also, functionally-designed

systems are sometimes difficult to modify due to process interdependencies, and alternate design decomposition are rarely considered.

4.2 Object-Oriented Design

The object-oriented approach uses top-down techniques for identifying self-contained objects, and bottom-up techniques for synthesizing them into a finished system. Booch [1986] describes the following object-oriented design steps:

> *1. Identify objects*[4] *and their attributes.* For large projects, an analysis technique such as SSA or SADT is useful for identifying objects; for smaller systems, objects and attributes can be determined through nouns and adjectives in the requirements definition.
>
> *2. Identify the operations suffered or performed by each object.* Objects can be **actors**, which perform operations; **servers**, which suffer operations; or **agents**, which do both. Operation primitives are either **constructors**, which alter an object's state, **selectors**, which inspect/evaluate an object's current state, or **iterators**, which visit every part of an object.
>
> *3. Establish the visibility relationships of objects.* A hierarchy of objects is established by determining object inputs and their sources.
>
> *4. Establish the interface of each object.* Object exports are specified.
>
> *5. Implement each object.* The form of object exports is specified, and an architectural framework for the overall system is developed.

Object-oriented designs eliminate global data since communication between objects is done solely through message passing. Such systems are therefore easy to maintain, since modifications to one object will not have unexpected side effects on another. Further, objects may execute either in sequence or in parallel, so real-time processes can be precisely specified. However, the object-oriented design process lacks an overall system design abstraction to keep the objects converging upon the original specification. Also, some types of systems are not well-suited to object-oriented design because objects are difficult to define

---

[4] An object may be defined as "an entity with a state and a defined set of operations to access and modify that state" [from Sommerville, p. 204].

and state information is unimportant. Like functionally-designed systems, object-oriented designs rarely consider alternate design decompositions.

4.3 Formal Design

Formal specifications required by the formal transformation process cannot be generated until the requirements definition/specification and architectural design have been completed. The formal design process, including a feedback loop for detecting and correcting errors, omissions, and inconsistencies, is shown in Figure 1 [Sommerville, p. 124]. This illustration shows that an architectural design must be developed through the phases of requirements and design specification before the precise specification can be produced. It should be noted that the process of precise specification may lead to corrections of errors, omissions, and inconsistencies that were made in previous steps, thus using the feedback loop.

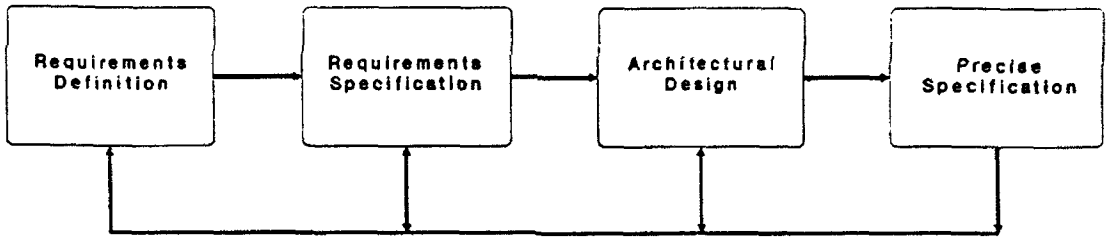


Figure 1 - The Formal Specification Development Process

A formal specification represents the system as a set of functions. Each is specified by name, parameters, and pre- and post-conditions using arithmetic operators and set notation. The following steps may be used to develop these specifications [Sommerville, p. 130]:

1. Define the function in terms of name, parameters with variable types, and return variable type.

2. Establish pre- and post-conditions by:

    2a. Establishing and expressing the range of input parameters over which the function is intended to behave correctly, and express their constraints as a predicate.

    2b. Specifying the output condition that must hold for the function (if it behaves correctly) as a predicate.

    2c. Establishing and specifying the changes (if any) to the function's input parameters.

Formal specifications have the advantage of being automatically transformable into prototype systems for testing and evaluation. However, as was noted earlier, this is a relatively new and unfamiliar technique, and some classes of systems may be difficult to specify formally.

None of these design methods provide a satisfactory solution to the software dilemma, but they do appear to complement each other. Object-oriented design may possibly be used as an enhancement to functional design: functional designs can be derived from system requirements, and then translated into object-oriented designs as necessary to represent concurrent parallel and serial processes and state information. Such designs can then be implemented conventionally or translated into a formal specification and transformed into executable code. Hopefully formal design transformation methods will continue to be refined, since this methodology shows promise in remedying the problems of system enhancement and modification.

The previous review of software processes and requirements and design specifications leads to some observations about current software engineering practices.

1. Most software processes are requirements-driven rather than objectives-driven: the primary emphasis is on the immediately-required functionality rather than organizational objectives or goals.

2. Most software systems are not designed with the concept of system modification in mind.

3. Most methods do not emphasize the consideration of design alternatives or document why alternatives are chosen.

To address these issues, a design method not traditionally used for software engineering called Knowledge-Based System Design will be applied to software system design.

5.0 A Knowledge-Based Software Design Process

Knowledge-Based Design is a simulation-based design method that approaches the design process from an objectives-driven, multiple-alternative perspective. It follows a conventional design approach with well-defined phases and transition milestones. Its knowledge based approach to configuration selection provides alternate design configurations that can be evaluated. Development of a software process based on this methodology will consist of an overview of the original design methodology, definition of the phases of the methodology as adapted to software design, and presentation of an example system design using the modified process.

5.1 Overview of Knowledge-Based System Design [Rozenblit and Zeigler, 1988; Rozenblit and Huang, 1990]

Key concepts of Knowledge-Based System Design include system entity structure modelling and development of a generic experimental frame. The **system entity structure** (SES) is a tree-like graph used to specify possible design configurations [Zeigler, 1984]. *Entities* represent system components, *aspects* represent entity decompositions, and *specializations* show entity classifications. Entities and aspects/specializations are placed alternately within the entity structure.

A **generic experimental frame** consists of variables that express performance parameters that determine the system configuration [Rozenblit and Huang, 1987]. Such frames are used to prune the SES by selecting among the various aspects and specializations and generating one or more possible design configurations that correspond to the design objectives. A knowledge base consisting of production rules is developed to select among components within the SES: **selection** rules resolve entity choices within specializations, based on system constraints, and **synthesis** rules select components within decompositions,

and ensure that aspect and specialization selections are configurable. Following the pruning process, a model composition tree is generated that represents a set of hierarchically-ordered model components that may be coupled together and evaluated.

To summarize, Knowledge-Based System Design consists of the following phases:

1. *Identify goals and objectives.*

2. *Specify requirements and constraints.*

3. *Model the system.* This is accomplished by representing various system classifications and decompositions in a system entity structure.

4. *Generate possible solutions.* A generic experimental frame based on system constraints is developed to identify one or more possible system configurations.

5. *Develop a model or models based on the identified system configurations.*

6. *Evaluate the alternatives through simulation.*

An adaptation of this methodology to software design will now be presented.

5.2 Adaptation to Software Design: Software Process Phases

The proposed software process involves the creation, pruning, and completion of the general computer system SES shown in Figure 2. The system should be modelled to include hardware as well as software, because some systems require hardware to be selected and configured, and hardware and software design should be considered concurrently. For systems with given hardware platforms, the hardware branch of the SES is pruned, and hardware components appear as experimental frame variables that influence the functional and non-functional requirements.

This model decomposes the software process into requirements specification, design specification, and implementation. Requirements specification consists of functional and non-functional requirements, and design specification consists of pruned functional requirements and programs. Programs are decomposed into processes and data. Implementation aspects include creation of programs and data files.

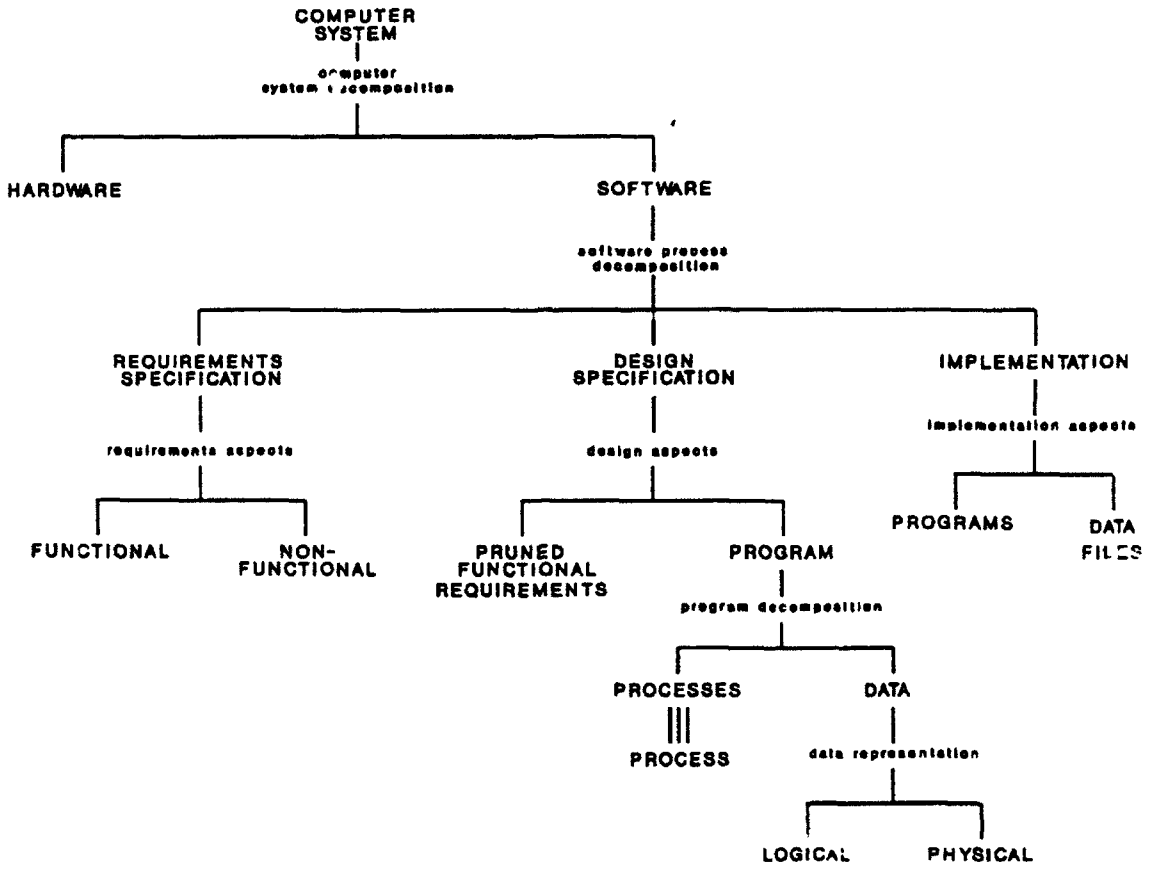


Figure 2 - High-Level System SES

The elaboration of this diagram into a complete system model will be accomplished through a series of phases. Definitions and justifications of the phases are as follows:

5.2.1 Identify goals and objectives

Goals consider long-term, high-level aspirations to be achieved by the system: e.g. "To increase office productivity and company profits." *Objectives* are specific ways of achieving goals: e.g. "Build a computerized system to efficiently handle our accounting processes." A goal can have several objectives attached to it; for example, building a computer system may serve to increase office productivity, but other objectives, such as decreasing office personnel, may have to be realized before the goal of increasing company profits will be achieved.

This approach encourages an organizational view of the software system, and allows system designers to consider the system from an objectives-driven, broadly-focused, multiple solution viewpoint. Goals and objectives are specified as textual lists, with objectives as sub-topics under goals.

5.2.2 Define requirements and constraints

System functional and non-functional requirements and constraints are developed for objectives that require design and implementation of computer systems.

5.2.3 Model system requirements specifications

This phase requires developing the Requirements Specification branch of the System SES. The non-functional requirements are decomposed into the categories suggested by Roman [1985] *(interface, performance, operating, life-cycle, economic, and political)*, and shown in Figure 3. These represent areas of potential system design decisions. Non-functional decisions and alternatives should be modelled as SES specializations, with their influencing constraints included as experimental frame variable types. The selected alternatives, which will be called design constraints, will influence the functional design.

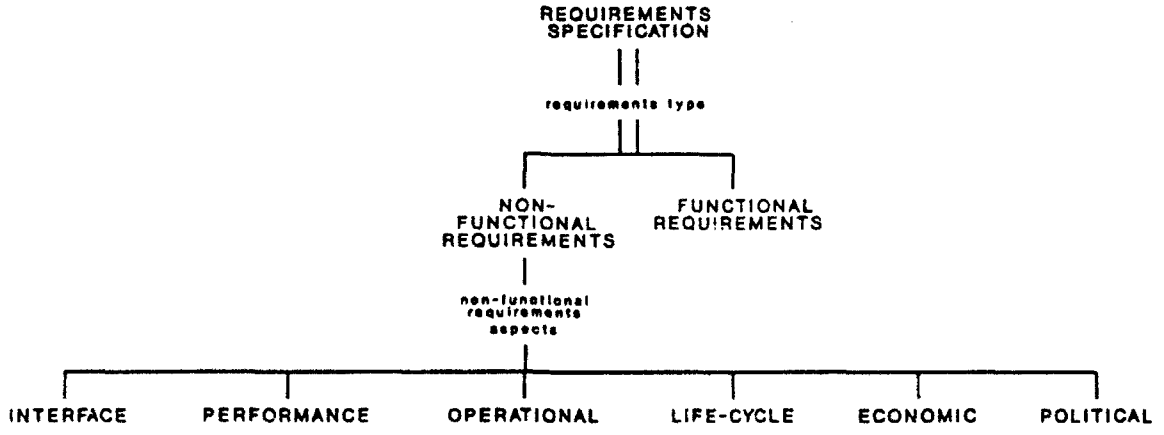


Figure 3 - System Non-Functional Requirements Decomposition

Functional requirements are entered into the SES according to the functional requirements list previously defined. Alternate ways of achieving these requirements are then proposed, and system and design constraints that may influence these alternatives are included as variable types. This approach encourages consideration of creative, non-obvious solutions from an objectives-driven approach. Alternatives are considered that may be beyond the current specifications or constraints, but such alternatives may be used for later enhancements, or may be incorporated within the current specification by altering constraints in an acceptable manner.

5.2.4 Generate possible configurations

The requirements branch of the SES is now pruned to generate design constraints and one or more possible design configurations. A knowledge base is developed to capture the relationships between system constraints, design constraints identified through non-functional requirements, and functional requirements. Selections of specializations in the SES are determined through *selection* rules using a backward-chaining inferencing process; selection sets within decompositions in the SES are determined using *synthesis* rules through a forward-chaining process. A final synthesis step validates the viability of the overall configuration. This is accomplished using a specially-created inferencing system developed in PC-Scheme that prompts the user for constraint information, and then presents recommended design configuration depending on selected constraints. More details on this system will be presented in the next section when an example is presented.

This phase provides a functional requirements specification from which a system model can be derived and evaluated. The knowledge base also documents the system constraint values that determined design constraints and ultimately generated the selected design configuration. Alternate configurations can be generated by manipulating system constraints with unknown or unsure values, and if constraints should change or decisions prove to be non-optimal, backtracking can be used to modify the system configuration.

5.2.5 Develop a design model

The pruned functional requirements are now condensed into a system model with functional requirements specified according to the design decisions generated in the previous phase. This model is examined and evaluated, and areas of uncertainty about the workability of certain software implementation techniques with the given hardware configuration may be resolved through simulation.

5.2.6 Develop a design specification

The tested design model is now transferred to the design branch on the SES and translated into processes and data. This is accomplished through stepwise refinement: as processes are identified and refined, logical and physical data requirements are defined also.

5.2.7 Implement the system

The design specification processes and physical data branches are transferred to the Implementation branch of the SES, and translated into executable code and data files. As more design decisions are encountered or if past decisions are changed, the knowledge base is updated.

These phases will be illustrated and developed through an example.

6. Design and Implementation of a Distributed Electronic Meeting System

6.1 System Overview

A significant research effort exists within the University of Arizona's MIS department to develop software and investigate group processes for electronically-supported meetings. So far, this research has focused on "same-time/same-place" groupware: meeting participants use networked workstations in a large conference room and communicate either verbally or electronically through software tools that support functions such as brainstorming, issue analysis, and voting [Saffo, 1990].

A current research project considers using the process in different participant time/spaces frames. One phase of this project is to develop a group support system that operates on local or wide area networks with participants distributed in different places and possibly working at different times. The design of this software will be used to demonstrate the application of Knowledge-Based System Design to software design. The specification, design, and implementation of this system will be outlined in terms of the phases of this process.

6.2 Software Process Phases

6.2.1 Goals and Objectives

Goals and objectives are presented for the entire research project rather than just the software development phase in order to give a global perspective to system requirements and constraints.

GOAL 1. Determine the requirements of a software system that supports group tasks where users are distributed in space and/or time.

OBJECTIVE 1.1. Design and implement a prototype group support system based on the existing Arizona GroupSystems tools, but with added features to make it suitable for distributed time/space environments.

OBJECTIVE 1.2. Observe and evaluate the system through beta-testing, laboratory experiments, and field studies/experiments to refine the system's user requirements.

OBJECTIVE 1.3. Modify the system as necessary.

GOAL 2. Determine if the use of a distributed electronic meeting support system is beneficial to work groups.

OBJECTIVE 2.1. Run a series of laboratory experiments with user space/time/support configuration as the independent variable, and task efficiency, effectiveness, and user satisfaction as the dependent variable.

OBJECTIVE 2.2. Evaluate the experimental results and attempt to draw conclusions about the benefits of distributed electronic meeting support software.

6.2.2 Requirements and Constraints

System requirements and constraints as related to the software system identified in Objective 1.1 are defined.

Functional Requirements:

1. Idea generation, issue analysis, and voting tools for users working synchronously or asynchronously over either a LAN or WAN.

2. Communication channels among group members and the group leader.

3. Methods of preserving group dynamics (visually and/or aurally).

4. Group status and activity monitoring facilities.

5. Session initiation capability for all group members.

6. Concurrent access to other software tools.

7. The ability to easily monitor the number of times a participant accesses the group status, messaging, and concurrent access functions (to determine if these are necessary or desirable features).

8. The ability to easily disable the group status, messaging, and concurrent access functions for experimental manipulations.

Non-Functional Requirements:

1. Intuitive, consistent, professional-quality interfaces with meaningful error messages.

2. Easily portable to different operating systems and networks.

3. Expandable to allow addition of new group processes (e.g. policy formation, stakeholder identification).

4. Well-documented and easy to modify.

5. Reliable.

Present System Constraints:

1. Prototype time frame = 6 months.

2. No hardware, software or other equipment purchases.

3. Hardware constraints: CPU: Intel 8088
Main Memory: 640K
Disk: 20 MB fixed
Network: Novell
Input: keyboard and/or mouse
Monitor: Monochrome or color

6.2.3 System Requirements Specifications

The non-functional and functional requirements branches of the SES are now expanded to identify non-functional system design issues and functional modules. The expansions of these two branches are considered separately.

Non-functional Requirements. This expansion identifies system design decisions of a non-functional nature as related to non-functional requirements and design constraints. The expansion is illustrated in Figure 4 and explained below.

Interface. The decision to include a text or graphical interface was determined to be based on the user's CPU type, the presence of a dedicated graphics co-processor, and the required system response time. It was decided that if response time was crucial, then a graphical interface would only be used if a Motorola 68000 or 68030 CPU or a graphics co-processor was present. Otherwise, a text interface will be used.

Performance. Two decisions were identified here: the structure of the system data files, and the network communication path. Data can be structured either as a normalized database or as flat text files. A normalized database is desirable for systems with large amounts of data, but is costly in terms of system response time, data throughput, and programmer-hours and software cost. Flat files are satisfactory for systems with limited data volumes.

Network communication can be achieved either through file input/output on the central server, the network BIOS, or both. File i/o is inherently more reliable and portable than network BIOS communication, but provides decreased network throughput. A combination of these paths could be used by communicating the most important data through file i/o, and less critical but more time-sensitive data, such as user participation level data, through the BIOS.

Operational. An operational consideration concerns whether data and executable program files are stored on the central server or on the individual user machines. Related constraints include user disk size, system security, data file size, and network traffic. If system security

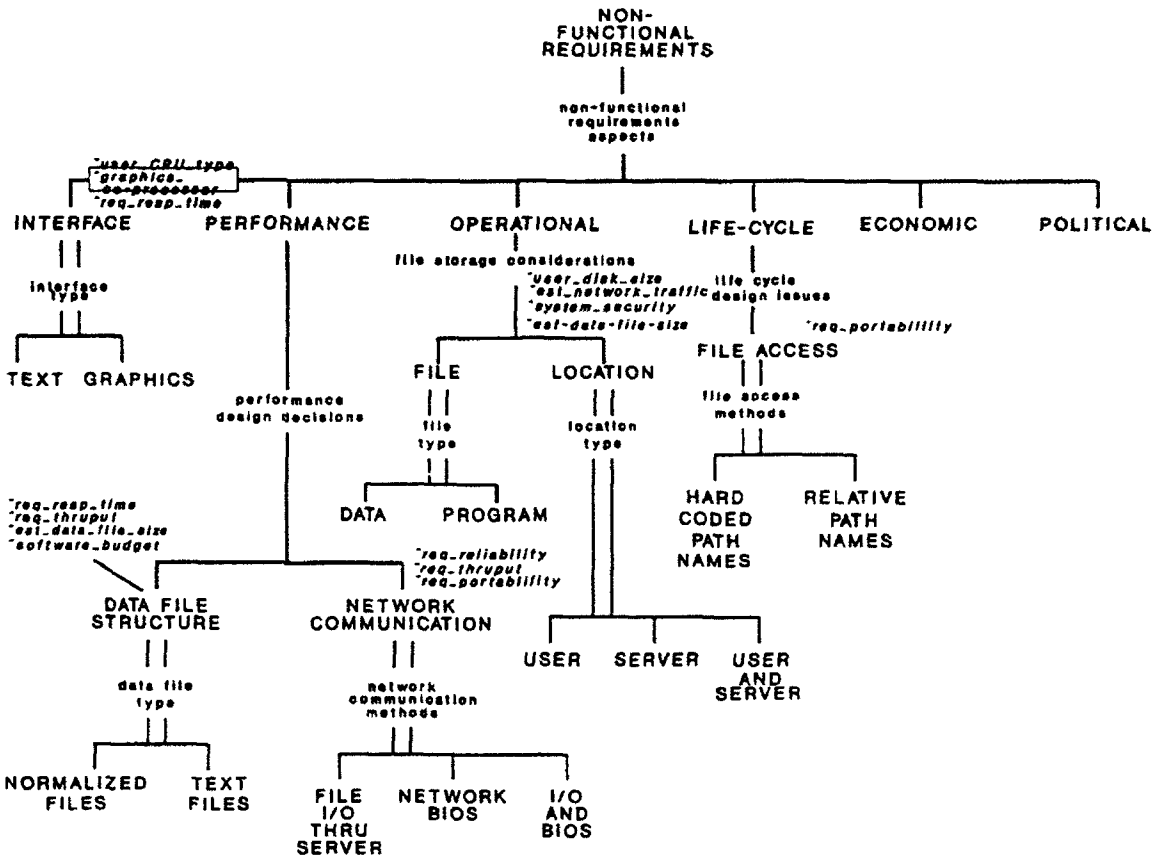


Figure 4 - Project Non-Functional Requirements

is vital or if users have little disk storage space, then all files should remain on the server. If data files are large or if network traffic is heavy, then some files should be on the user machines.

Economic, Political. No economically- or politically-motivated design decisions were foreseen.

Functional Requirements. The functional requirements branch is decomposed into the system functional requirements previously defined. These requirements are then decomposed into functional aspects. Selection among these aspects is primarily driven by development time and budget. The functional requirements expansion is illustrated in Figure 5 and explained below.

Group Processes. Group processes are decomposed into **idea generation**, **issue analysis**, and **voting**. **Idea generation** can be divided into separate processes where group members

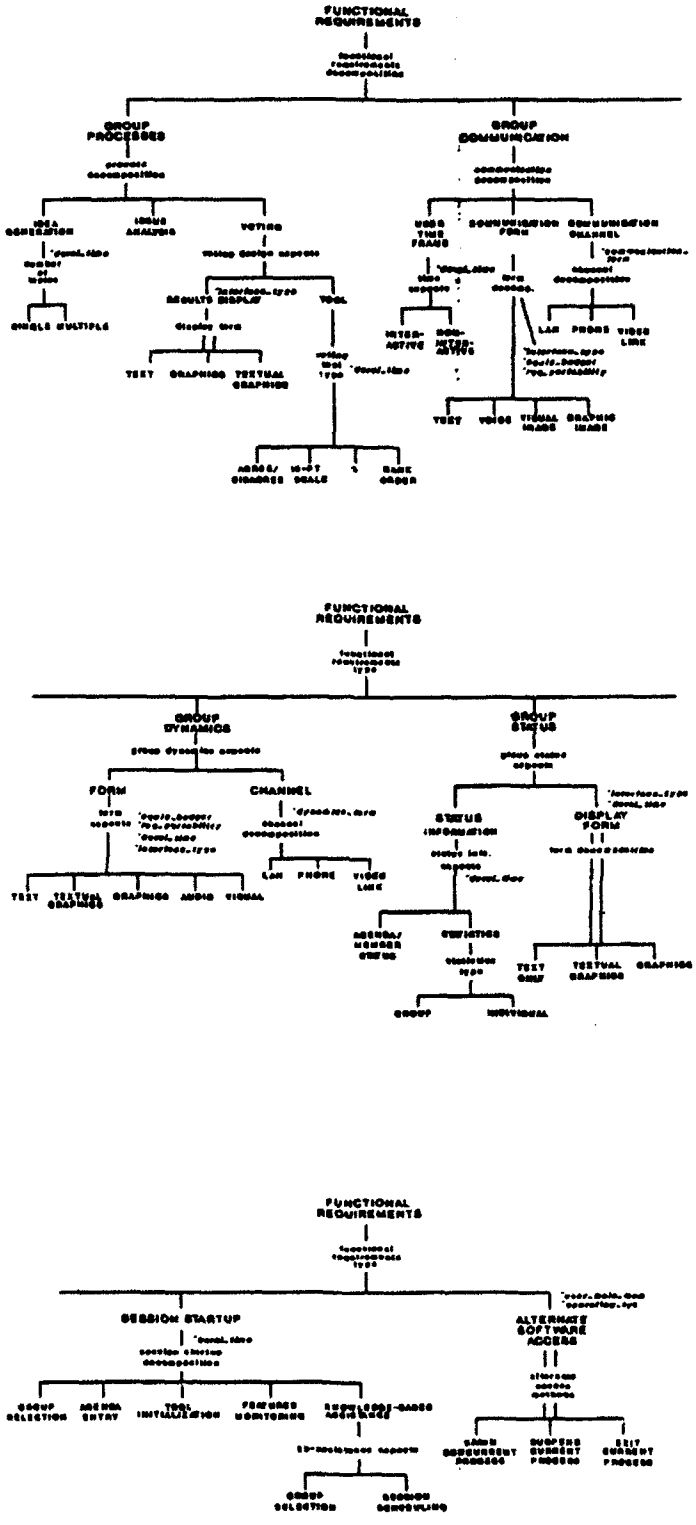


Figure 5 - DEMAS System Functional Requirements

may enter ideas on either a single topic or on multiple topics. **Voting** may be accomplished in a number of ways: agree/disagree, 10-point scale, percentages, multiple choice, and rank ordering. Voting results may be displayed in text, graphics, or textual graphics, depending on the type of interface used. The selection of group processes to be implemented is solely a factor of development time.

Group Communication. Group Communication can be decomposed into **User Time Frame, Communication Form,** and **Communication Channel.** The **User Time Frame** may be *Interactive, Non-Interactive* or both. The **Communication Form** may be decomposed into *Text, Voice, Visual Image* (where you see the person with whom you are communicating), or *Graphic Image*. The configuration selection is based on the interface type, equipment budget and desired system portability. The **Communication Channel** may be either a *LAN, Phone, Video Link*, or combination of these, depending on the communication form.

Group Dynamics. Group Dynamics design must consider the presentation form of participant interaction. Information may be *Text, Textual Graphics, Graphics, Audio,* or *Visual*, depending on the equipment budget, desired system portability, development time, and interface type. Depending on the form chosen, the data may be transmitted over a *LAN, Phone, Video Link*, or a combination of these. Visual images and phone/video links are limited by the equipment budget; graphic images are constrained by the interface type.

Group Status. Group Status considerations include what status information is to be displayed, and how it will be presented. Depending on development time, possibilities include *Agenda Status, Member Status*, and *Group and/or Individual Statistics*. The **Display Form** can be *Text Only, Textual Graphics,* or *Graphics*, and is constrained by development time and the selected interface type.

Session Startup. Session Startup may be comprised of **Group Selection, Agenda Entry, Tool Initialization, Knowledge-Based Assistance,** and **System Function/Monitoring Setup,** depending again on development time. Group Selection and Agenda Specification could potentially be integrated with a knowledge base/database so the session originator could easily retrieve information about group and/or group members' areas of expertise, interest, etc., based on criteria such as user profiles and past session participation. Further, scheduling of task times could be integrated with potential group members' electronic calendars to determine possible session times. System Function Setup allows the session initiator to monitor participant accesses to system features and disable system features for experimental purposes.

Alternate Software Access. Alternate Software Access can be achieved in three ways: by spawning a concurrent process, suspending the current process and starting another, or by exiting the current process and starting another. Spawning a concurrent process requires a large amount of user main memory and an operating system that supports concurrent processes, such as UNIX or OS/2. Suspending the current process and starting another is

simpler, only requiring enough main memory to store the state variables of the suspended process. Exiting the current process is the simplest but slowest method, with state variables stored on disk.

6.2.4 Possible System Configurations

The requirements specifications SES must now be pruned to derive a requirements configuration that can be translated into a design specification. Since no available systems provided the required flexibility in inferencing patterns and the ability to easily create complex but easily-understood rules, a combination backward/forward-chaining inference engine and corresponding rule base was created in PC-Scheme. A backward-chaining rule base was created to instantiate all required design constraints (e.g. specializations). After this is completed, the forward-chaining inferencing system determines the component configurations based on the design constraints and confirms the viability of the selected configuration. This system also allows rule actions to include list manipulations, and defines values that may be directly incorporated into DEVS-Scheme simulation models. Source code for the inference engine and knowledge base and transcripts of sample consultation sessions are included in the Appendix.

Three separate configurations were developed: one uses present system constraints; another investigates the configuration in a medium-range development period; the third explores an increased user base over a medium-range development period. The user-provided system constraints, inferred design constraints, and functional requirements configurations are summarized in Table 1. Areas where constraints or configurations vary are bolded and highlighted, and summarized as follows:

1. Increased development time suggests implementation of more system functions and features, such as Interactive Communication, Textual Graphics Displays, and numerous extra functions within Group Status and Session Startup.

2. Increased development time in conjunction with a larger user base causes an interesting dilemma in terms of system security. Since program and/or data files will probably have to be stored on the user machines, encryption or other security measures will have to be considered.

CONSTRAINT VARIABLES	Case 1: Current Configuration	Case 2: > Development Time	Case 3: > User Base & > Development Time
Graph. Co-Processor	No	No	No
User-CPU-Type	8088	8088	8088
User Main Mem.	=< 2 MB	=< 2MB	=< 2MB
User Disk Size	20-40 MB	20-40MB	20-40MB
Operating System	MS-DOS	MS-DOS	MS-DOS
Req-Resp-Time	High	High	High
Req-Thruput	High	High	High
Req-Reliability	High	High	High
Req-Portability	High	High	High
System Security	High	High	High
Program File Size	Small	Small	Medium
Data File Size	Small	Small	Medium
Network Traffic	Low	Low	Medium
Software Budget	None	None	None
Equipment Budget	None	None	None
Development Time	Short	Medium	Medium
INFERRED VARIABLES			
Interface Type	TEXT	TEXT	TEXT
Data File Type	TEXT FILES	TEXT FILES	NORMALIZED FILES
Net. Comm. Path	COMB. IO/BIOS	COMB. IO/BIOS	COMB. IO/BIOS
Prog. File. Loc.	SERVER	SERVER	USER-AND-SERVER (w/encryption)
Data File Loc.	SERVER	SERVER	USER-AND-SERVER (w/encryption)
File Access Strategy	RELATIVE PATHS	RELATIVE PATHS	RELATIVE PATHS
Idea Gen. Tool	Single-Topic	Single Topic, Multiple Topic	Single Topic, Multiple Topic
Voting Tool Type	Rank-Order	Rank Order, Agree/Disagree	Rank Order, Agree/Disagree
Voting Display	Text	Text, Textual Graphics	Text, Textual Graphics
Comm. Time Frame	Non-Interactive	Interactive, Non-Interactive	Interactive, Non-Interactive
Comm. Form	TEXT	TEXT	TEXT
Comm. Channel	LAN	LAN	LAN
Dynamics Form	TEXT	Text, Textual Graphics	Text, Textual Graphics
Dynamics Channel	LAN	LAN	LAN
Status Info.	Agenda, Member Status	Agenda, Member Status, Group Statistics	Agenda, Member Status, Group Statistics
Startup Info.	Group Selection, Agenda Entry, Tool Initialization, Features Monitoring	Group Selection, Agenda Entry, Tool Initialization, Features Monitoring, Participant Knowledge	Group Selection, Agenda Entry, Tool Initialization, Features Monitoring, Participant Knowledge
Alt. Access	Exit Current Process	Exit Current Process	Exit Current Process

Table 1 - Alternate Design Configurations

The system entity structures showing pruned functional and non-functional requirements for the current system configuration are included in the Appendix. The evaluation of these functional requirements and development of an initial design specification is now performed.

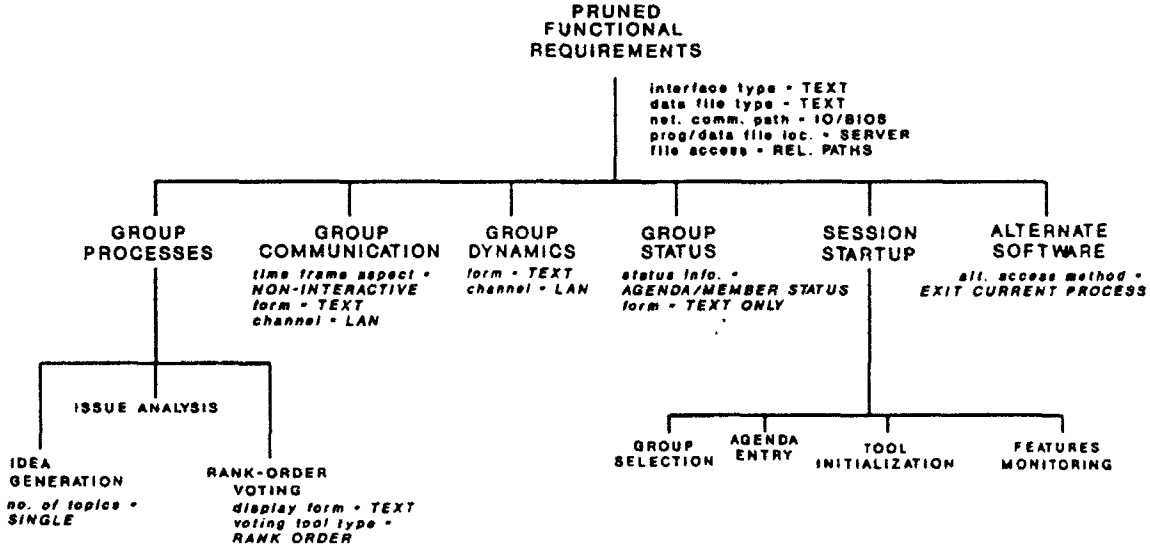


Figure 6 - Project Pruned Functional Requirements

6.2.5 Design Model

The initial Pruned Functional Requirements (Figure 6) shows the pruned processes, with general design constraints applicable to all functions shown at the top, and process-specific design constraints written below the corresponding process. At this point, it was realized that the network communication path for specific processes would have to be specified. We tentatively decided to achieve network communication within the Idea Generation Tool through server i/o. However, the question arose as to whether communication solely through file exchange on the network server would provide the required high user response rate. This tool will require users to exchange files at a fairly high rate; it is desirable to determine the effect of the number of users on the given network/server configuration on system response. A design model was developed using the distributed experimental frame concept to simulate this portion of the system. The desired result of the simulation is average system response rate for each user and for the overall

system; the number of users in the system will be manipulated to see how the performance of the system reacts.

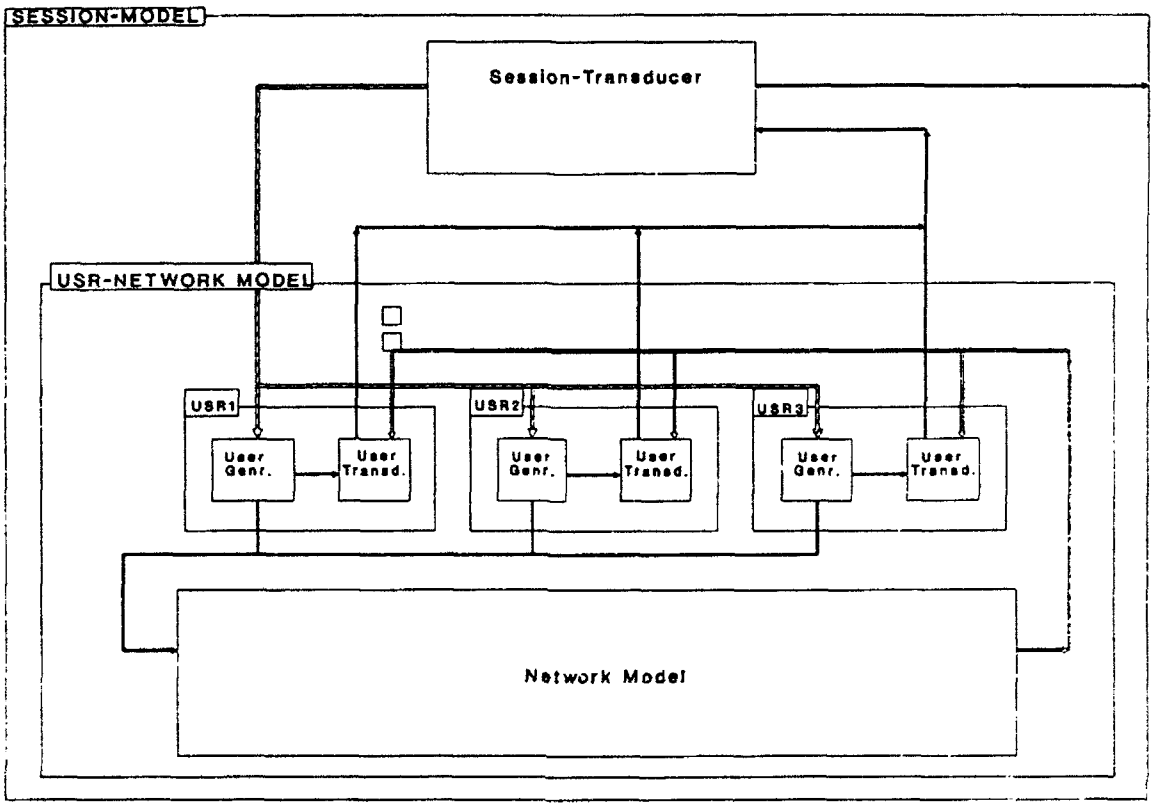


Figure 7 - Proposed Simulation Model

The proposed simulation model is shown in Figure 7. System components and their functions are as follows:

Session-Transducer. This component controls the simulation by sending a start signal to the user generators at the beginning of the simulation and a stop signal to them after the desired observation interval. It also receives all the local average job response rates from the user transducers, and compiles them into a system average.

User-Genr. Upon receiving the start signal from the Session-Transducer, this component generates random JOB-IDs at random intervals compatible with average user task-generation rates, and submits them to the local User-Transd and to the Network-Model (with the user-number appended). When a stop signal is received, a stop signal is sent to the User-Transd, and no more jobs are generated.

User-Transd. When a JOB-ID is received from the local generator, the User-Transd notes the local clock time. When a JOB-ID is received from the Network-Model, the processing time of the job is calculated, as well as the average job response rate for all jobs. When a stop signal is received from the local generator, the User-Transd submits the local average job response rate to the Session-Transducer.

Network-Model. If the Network-Model is busy when a JOB-ID/USR# is received, the JOB-ID/USR# is placed in a first-in, first-out queue. Otherwise, the job is processed according to the given "average network file processing time", and then output to the user. As soon as a job is output, the queue is checked to see if another job has arrived; if it has, that job is processed.

The formal DEVS specifications for these components are included in the Appendix. A simulation was not performed due to the short time frame and prototype nature of the software project. However, this model illustrates how simulation can be applied to the evaluation of software system design. It may be implemented in the future to determine necessary hardware parameters such as required network communication rate and server i/o rate for a given user base if dedicated hardware is purchased in the future.

6.2.6 Design Specification

The Group Status function will be used to illustrate the design phase, since it is fairly simple, but illustrates both process and data design concepts. The process decomposition SES is presented in Figures 8 and 9, and the data decomposition in Figure 10. Group Status is comprised of two functional aspects: AGENDA and MEMBER STATUS. The desired outputs of these processes are specified, and each process is then decomposed into sub-processes until each sub-process can easily and unambiguously be translated into code.

The logical and physical data design of the data files related to these modules was an important component with the design these functions. The logical data structure was considered first, and it was determined that the relevant entities to these processes were SESSIONS and USERS. These are decomposed using the SES property of multiple decomposition into a single SESSION and USER. Every data item related to SESSIONS in this context was identified; these were NAME and TOOLS. Another multiple decomposition was performed to obtain TOOL, data items related to the combined SESSION/TOOL entity were determined to be NAME, START DATE/TIME, STOP DATE/TIME, and TYPE. (A tool type is either synchronous or asynchronous).

Relevant USER data includes his/her NAME and SESSIONS. Following a multiple decomposition to SESSION, data aspects of the combined USER/SESSION entity include USER_STATUS (active or inactive), SESSION_NAME, CURRENT_TOOL, and LAST_LOGIN_TIME.

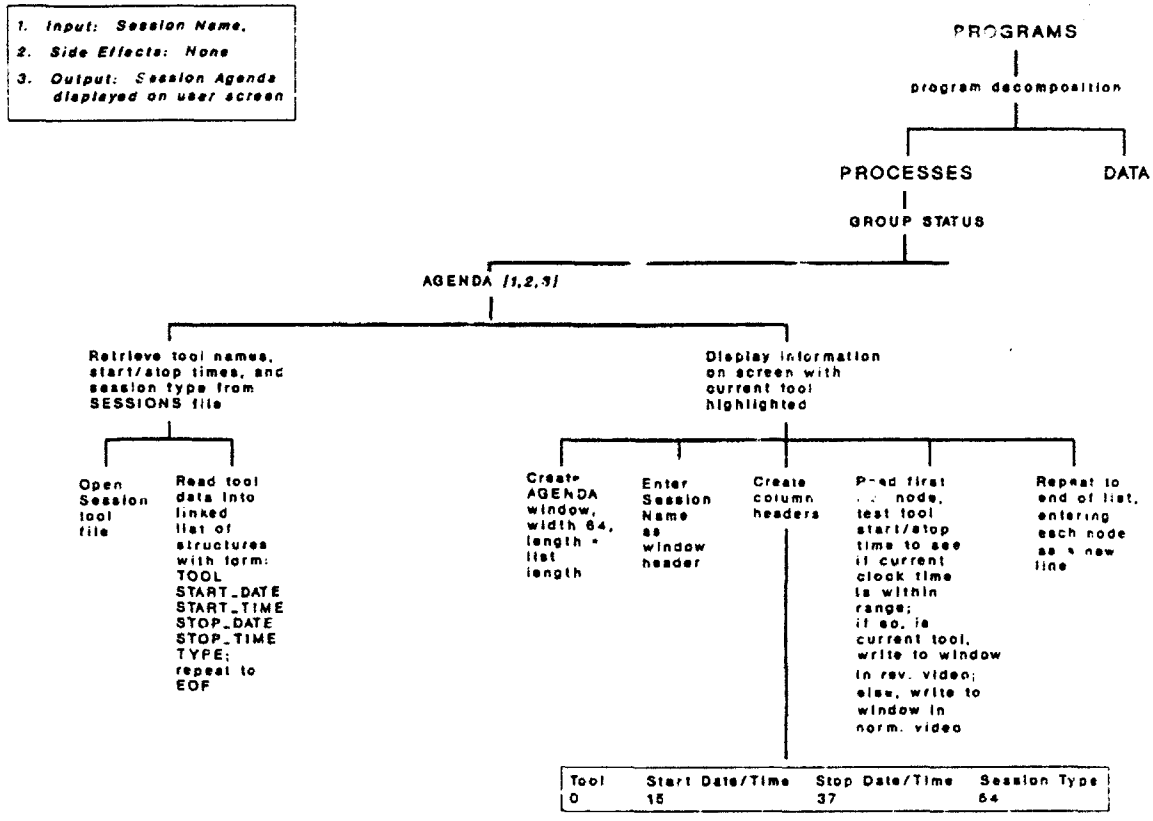


Figure 8 - System Process Decomposition (Agenda Module)

A possible physical structure for this data is shown in terms of both text files and normalized files. For the text file approach, a separate file is needed for each session to show its tools ("SESSION1_TOOLS"). Each tool is then listed by name, followed by its data. Similarly, a separate file is needed for each session to show its users ("SESSION1_USERS"), listing each user name followed by his/her data. This structure is clearly represented by the SES logical design. This is a potentially powerful design tool, since text files are often used to store data, but methods to aid in structuring such files are rare.

The normalized file design indicates two data files are required, one for each pair of multiple decompositions, with composite keys consisting of each multiple decomposition entity. Each record in the SESSION_TOOLS file can be accessed through the session name and tool name; likewise, every record in the USER_SESSION file is available through the user name and session name. To carry this concept further, it appears that a separate file will be necessary for every entity beneath an entity that has any unique aspects; if the lower entity is a multiple decomposition also, then a composite key consisting of a unique field from both the parent and child entity is required. The resulting database structure is in third

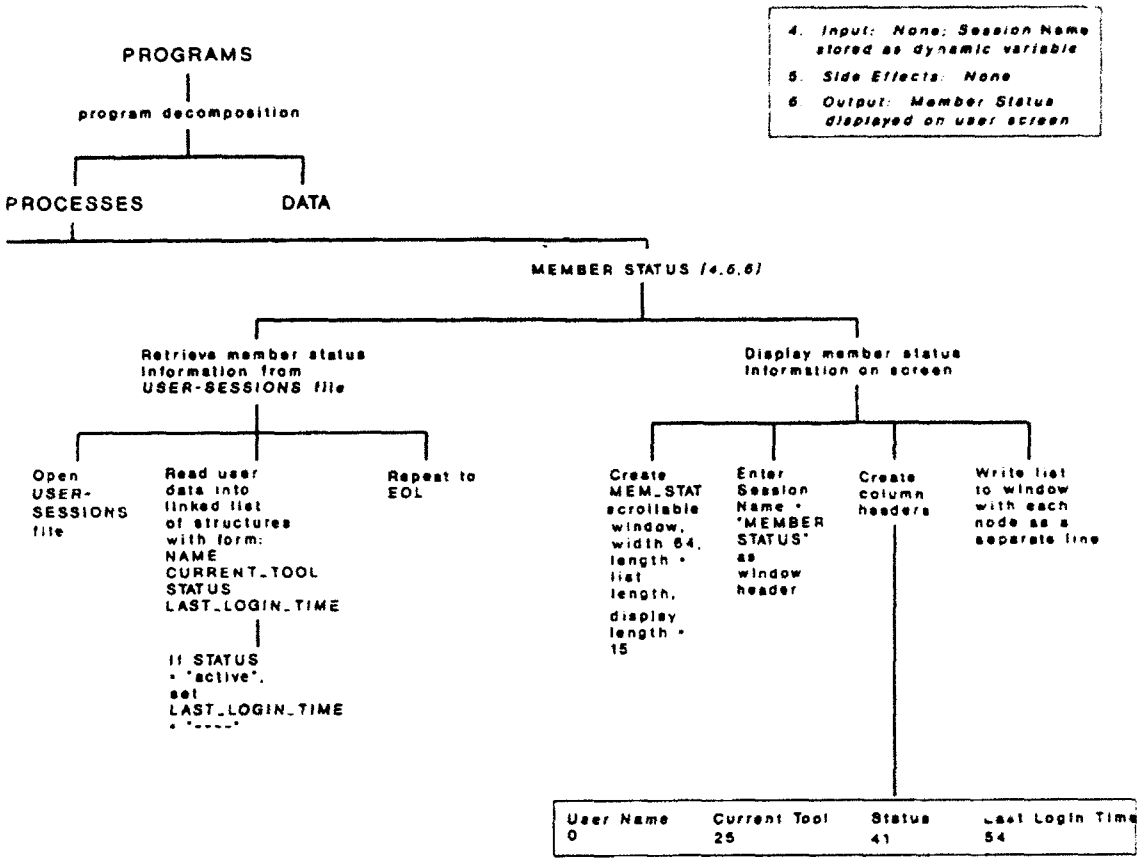


Figure 9 - System Process Decomposition SES (Member Status Module)

normal form, with every record having a unique primary key. If an entity has two child multiple-decomposition entities, and the same trio is present under both of the child entities somewhere, a ternary relationship is indicated. This is potentially a powerful way of representing a database schema and carrying it through to the physical design of the data files. While the concept of using the SES to represent a database schema has been previously proposed [Higa and Sheng, 1989], this method of combining data fields and related entities as decompositions within the SES seems more intuitive, and easily translates into data files.

6.2.7 System Implementation

The system is currently being implemented and is in a state of constant change, so no finished system structure charts or architecture diagrams are available.

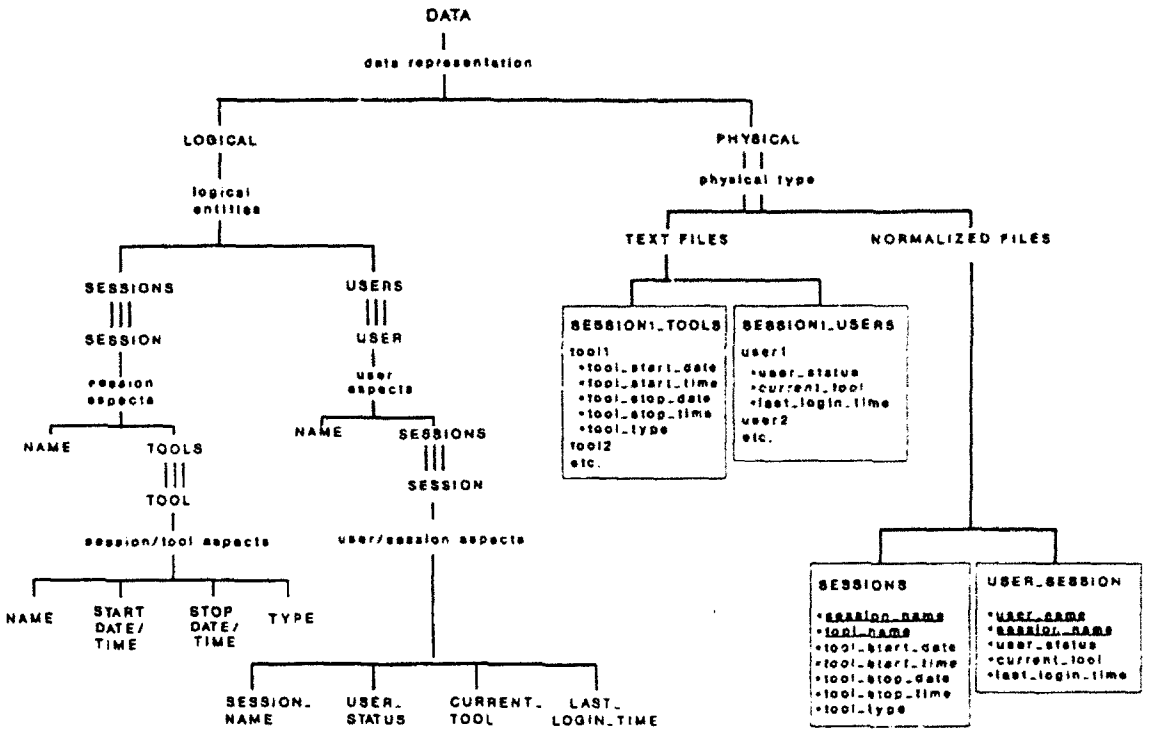


Figure 10 - Project Logical and Physical Data Design

7. Conclusions and Future Directions

This project provided the challenge of applying a general design methodology to a very specific and challenging domain. The methodology successfully addressed the issues of approaching software design from an objectives-driven viewpoint, forcing consideration of design alternatives and documentation of selection criteria, and designing systems with a long-term perspective with modification in mind. The design documentation generated from this project will serve as a high-level blueprint for future system development.

However, the project also reinforced the fact that software engineering is a very complex area, and there are no easy solutions. While this method works well for providing a high-level design specification, other methods must be incorporated or developed to specify lower-level programming specifications that can adequately model the complexity of the software.

The most interesting research ideas occurred in areas outside the main focus of the project. The development of an alternate SES pruning system in PC-Scheme may merit further consideration; and, further development of database design techniques using the SES should be investigated.

REFERENCES

Barstow, D. Artificial Intelligence and Software Engineering. *Proceedings of the 9th Int. Conf. on Software Engineering*, March 30 - April 2, 1987, Monterey, California, pp. 200-11.

Boehm, B.W. A Spiral Model of Software Development and Enhancement. *IEEE Computer 21*,5 (May 1988) 61-72.

Borgida, A., Greenspan, S., and Mylopoulos, J. Knowledge Representation as the Basis for Requirements Specifications. *IEEE Computer 18*,4 (April 1985) 82-91.

Booch, G. Object-oriented development. *IEEE Trans. Software Eng. SE-12*,2 (February, 1986) 211-221.

Brooks, F.P. Jr. *The Mythical Man-Month.* Addison-Wesley, Reading, Massachusetts, 1982.

Brooks, F.P. Jr. No Silver Bullet - Essence and Accidents of Software Engineering. *IEEE Computer*, April 1987, 10-19.

Cameron, J.R. An Overview of JSD. *IEEE Trans. Software Eng. SE-12*,2 (February 1986) 222-40.

Constantine, L.L. and Yourdon, E. *Structured Design.* Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

Couger, J.D., Colter, M.A., and Knapp, R.W. *Advanced System Development/Feasibility Techniques.* John Wiley & Sons, New York, 1982.

DeMarco, T. *Structured Analysis and System Specification.* Yourdon Inc., New York, 1978.

DeMarco, T. and Lister, T. *Peopleware: productive projects and teams.* Dorset House, 1987.

Gane, C. & Sarson, T. *Structured Systems Analysis: Tools and Techniques.* Prentice-Hall, Englewood Cliffs, N.J., 1979.

Higa, K. and Sheng, O.R.L. An Object-Oriented Methodology for End-User Logical Database Design: The Structured Entity Approach. In *Proceedings of COMPSAC '89* (Orlando, Florida, Sept. 1989).

Herninger, K.L. Specifying Software Requirements for Complex Systems: New Techniques and Their Application. *IEEE Trans. Software Eng. SE-6*,1 (January 1980) 2-13.

Parnas, D.D. and Clements, P.C. A Rational Design Process: How and Why to Fake It. *IEEE Trans. Software Eng. SE-12*,2 (February 1986) 251-57.

Ratcliff, B. *Software Engineering Principles and Methods.* Blackwell Scientific Publications, Oxford, UK, 1987.

Roman, G. A Taxonomy of Current Issues in Requirements Engineering. *IEEE Computer 18*,4 (April 1985) 15-22.

Ross, D.T. Structured Analysis (SA): A Language for Communicating Ideas. *IEEE Trans. Software Eng. SE-3*,1 (January 1977) 16-34.

Ross, D.T. and Schoman, K.E. Jr. Structured Analysis for Requirements Definition. *IEEE Trans. Software Eng. SE-3*,1 (January 1977) 6-15.

Rozenblit, J.W. A Conceptual Basis for Integrated, Model-Based System Design. Technical Report, National Science Foundation, January, 1986.

Rozenblit, J.W. and Huang, Y.M. Constraint-Driven Generation of Model Structures. In *Proc. of the 1987 Winter Simulation Conf.*, December, 1987, Atlanta, Georgia, pp. 604-11.

Rozenblit, J.W. and Zeigler, B.P. Design and Modeling Concepts. *International Encyclopedia of Robotics, Applications and Automation.* John Wiley & Sons, New York, 1988, pp. 308-22.

Saffo, P. Same-Time, Same-Place Groupware. *Personal Computing*, March 20, 1990, p. 57-58.

Sommerville, I. *Software Engineering.* Addison-Wesley, Workingham, England, 1989.

Trapnell, F.M. A systematic approach to the development of system programs. In *Proc. AFIPS SJCC 1969*, pp. 411-18.

Wirth, N. Program Development by Stepwise Refinement. *Communications of the ACM 14*,4 (April 1971) 221-227.

Zeigler, B.P. *Multifacetted Modelling and Discrete Event Simulation.* Academic Press, London, 1984.

Session-Transducer

T_s = $\{X, S, t_a, \delta_{int}, \delta_{ext}, \lambda, Y, S_o\}$

X = <AVTT>

S = (passive, start, stop, make-output)

t_a = $\{t_a(\text{passive}) = \infty\}$
$\{t_a(\text{start}) = 0\}$
$\{t_a(\text{stop}) = 0\}$
$\{t_a(\text{make-output}) = 0\}$

δ_{int} = {if observation-int = T, state = stop}
{if state(start, stop, make-output), state = passive}

δ_{ext} = ((AVTT, N) e, x)
if (passive, e, AVTT) update ΣAVTT/N, s = make-output
if ((stop,start,make-output), e, x) continue

λ = $\{\lambda(\text{stop}) = \text{stop-signal}\}$
$\{\lambda(\text{start}) = \text{start-signal}\}$
$\{\lambda(\text{make-output}) = \text{AVTT}, \Sigma\text{AVTT/N}\}$

Y = <stop-signal, start-signal, AVTT, ΣAVTT/N>

S_o = <ΣAVTT, N>

User-Genr

G_u = $\{X, S, t_a, \delta_{int}, \delta_{ext}, \lambda, Y, S_o\}$

X = <start-signal, stop-signal>

S = (passive, active, stop)

t_a = $\{t_a(\text{passive}) = \infty\}$
$\{t_a(\text{active}) = r_2\}$
$\{t_a(\text{stop}) = 0\}$

δ_{int} = $\{\text{state(active)} = (r_1, r_2) = (\Gamma(r_1), \Gamma(r_2))\}$
{state(stop) = passive}

δ_{ext} = if (passive, e, start-signal) output(r_1,r_2), state = active
if (passive, e, stop-signal) continue
if (active, e, start-signal) continue
if (active, e, stop-signal) state = stop
if (stop, e, x) continue

λ = $\{\lambda(\text{active}) = (r_1, (r_1, \text{usr-\#}))\}$
$\{\lambda(\text{stop}) = \text{stop-signal}\}$

Y = <r_1, (r_1, usr-#), stop-signal>

User-Transd

T_u = $\{X, S, t_a, \delta_{int}, \delta_{ext}, \lambda, Y, S_o\}$

X = $<r_1$, JOB-ID_{sol}, stop-signal>

S = (passive, stop)

t_a = $\{t_a$(passive) = $\infty\}$
$\{t_a$(stop) = 0$\}$

δ_{int} = {state(stop) = passive}

δ_{ext} = if (passive, e, r_1) update T_{arr}
if (passive, e, JOB-ID_{sol}) update T_{sol}, N, AVTT (= $\Sigma(T_{sol} - T_{arr})/N$)
if (passive, e, stop-signal) s=stop
if (stop, e, x) continue

λ = $\{\lambda$(stop) = AVTT$\}$

Y = <AVTT>

S_o = <T_{sol}, T_{arr}, N, AVTT>

Network-Model

N_m = $\{X, S, t_a, \delta_{int}, \delta_{ext}, \lambda, Y\}$

X = $<(r_1$, USR#)>

S = (passive, busy)

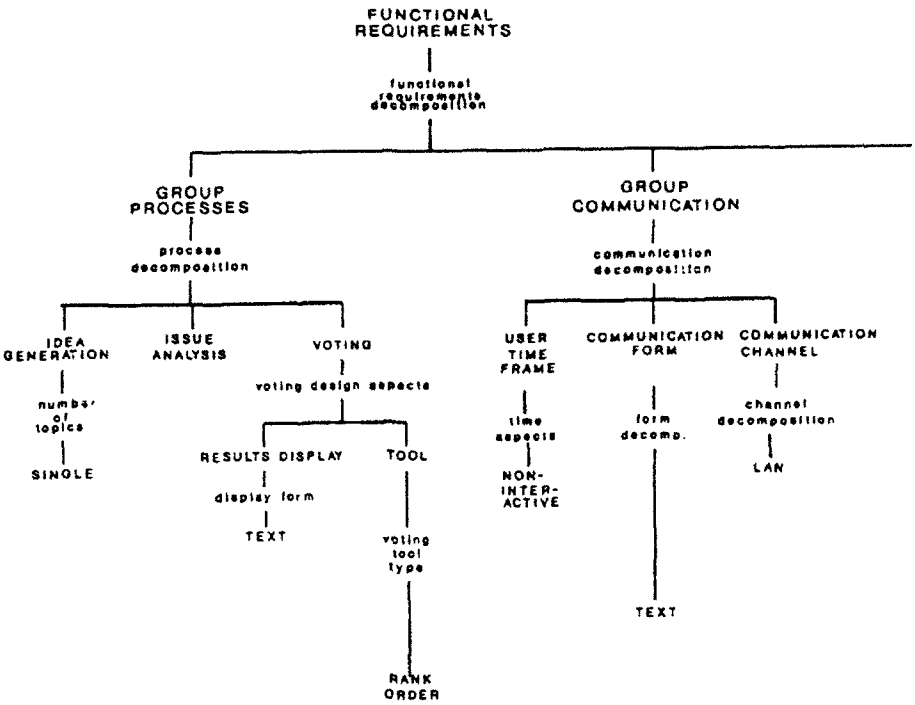
t_a = $\{t_a$(passive) = $\infty\}$
$\{t_a$(busy) = avg-proc-time$\}$

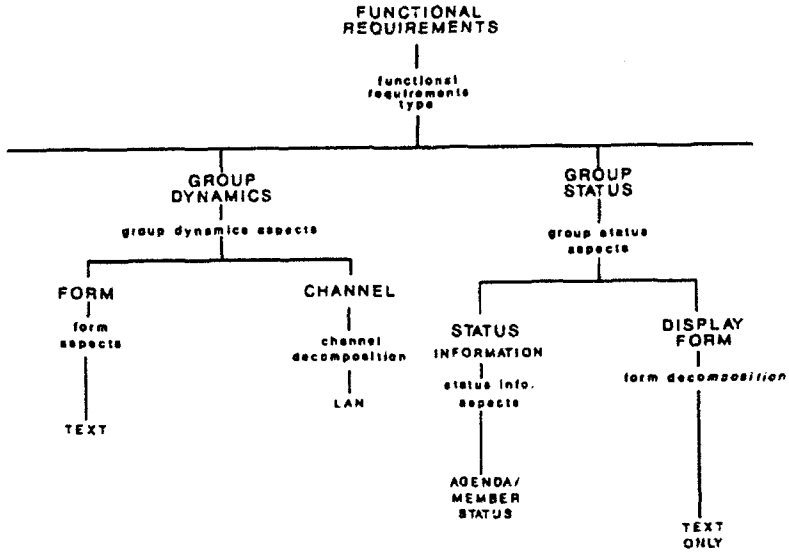
δ_{int} = {if queue(empty) state = passive}
{if queue (not empty) queue = rest of queue, state = busy}

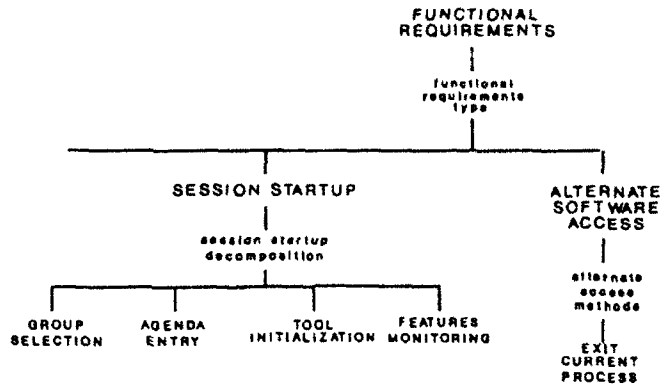
δ_{ext} = if (busy, e, x) continue
if (passive, e, x) s=busy

λ = $\{\lambda$(busy) = JOB-$ID_{sol}\}$

Y = <JOB-ID_{sol}>


FUNCTIONAL REQUIREMENTS
functional requirements decomposition
GROUP PROCESSES
process decomposition
IDEA GENERATION
number of topics
SINGLE
ISSUE ANALYSIS
VOTING
voting design aspects
RESULTS DISPLAY
display form
TEXT
TOOL
voting tool type
RANK ORDER
GROUP COMMUNICATION
communication decomposition
USER TIME FRAME
time aspects
NON-INTER-ACTIVE
COMMUNICATION FORM
form decomp.
TEXT
COMMUNICATION CHANNEL
channel decomposition
LAN


FUNCTIONAL REQUIREMENTS
functional requirements type
GROUP DYNAMICS
group dynamics aspects
FORM
form aspects
TEXT
CHANNEL
channel decomposition
LAN
GROUP STATUS
group status aspects
STATUS INFORMATION
status info. aspects
AGENDA/ MEMBER STATUS
DISPLAY FORM
form decomposition
TEXT ONLY


FUNCTIONAL REQUIREMENTS
functional requirements type
SESSION STARTUP
session startup decomposition
GROUP SELECTION
AGENDA ENTRY
TOOL INITIALIZATION
FEATURES MONITORING
ALTERNATE SOFTWARE ACCESS
alternate access methods
EXIT CURRENT PROCESS

Appendix IV

Software Review: NEXPERT OBJECT

Milam W. Aiken

Olivia R. Liu Sheng

Department of Management Information Systems

College of Business and Public Administration

University of Arizona Tucson, AZ 85721

602-621-2748

Abstract

NEXPERT OBJECT is an expert system shell developed by Neuron Data Inc. NEXPERT costs $5,000 for the microcomputer version, and a VAX version is available for $8,000. Neuron Data can be contacted in the USA at 444 High St., Palo Alto, CA 94301 (415-321-4488) or in the UK at 34 S. Molton, London, W1Y 1BP (441-408-2333). The version reviewed in this article was release 1.02 with Microsoft Windows. It was run on an AT&T 6386 microcomputer with 3 megabytes of RAM, an 80-megabyte hard disk, and a color monitor.

1 Introduction

A variety of sophisticated expert system shells have been introduced in the USA (M1, Exsys, Guru, PC Plus) and in Britain (Crystal, Leonardo, Savoir, and Xi+), but NEXPERT OBJECT presents many advantages over these competing products. NEXPERT's emphasis on an object-oriented approach makes it ideal for

large, complex knowledge bases as well as theoretical work in artificial intelligence. In addition, it allows developers to embed portions of its inference engine in their own code, combining the advantages of expert systems with algorithmic languages.

Another major advantage is its wide-spread interoperability. NEXPERT can run on a number of different machines including the IBM AT, PS/2, RT PC, 286, 386, Macintosh II and SE, Sun and Apollo workstations, VAX, and the IBM mainframe. It is compatible with many different operating systems including DOS, OS/2, Mac OS, Unix, VMS, and VM. This interoperability (NEXPERT is written in "C") ensures that user-developed knowledge bases will run on a variety of products as well as subsequent releases of NEXPERT.

NEXPET OBJECT is a full-featured expert system shell that is competitive with many higher-priced mainframe-based packages including Intellicorp's Knowledge Engineering Environment (KEE) and Inference Coporation's Automated Reasoning Tool (ART). Only a few of NEXPERT's many powerful features are detailed in this review.

2 Knowledge Representation

NEXPERT *is a hybrid system, combining rules and structured representations* known as objects.

2.1 Rules

A rule is the elementary chunk of knowledge in NEXPERT. It links facts or observations to assertions or actions. The basic structure of the rule is:

```
IF        condition THEN      hypothesis
   AND    condition     AND     actions
             .                   .
             .                   .
             .                   .
```

For example, to test for an alarm, NEXPERT might have the following rule in a knowledge base:

```
Rule Alarm_Test:
```

```
IF water_level_low THEN Alarm_is_Set
                        EXECUTE "alarm.exe" @string="alarm"
```

In the above rule, NEXPERT checks for the condition *water_level_low*. If the condition is true, it sets the hypothesis *Alarm_is_Set* to true and also executes a program to set an alarm, passing a program parameter in a string.

In addition, NEXPERT is integrated with several standard relational databases and spreasheets. The first level of integration is through transparent, built-in bridges to spreadsheet formats, such as NEXPERT's proprietary format NXP, *Sylk* (Microsoft Excel), and *WKS* (Lotus). This first level of integration also allows access to a number of databases including DBase III, DBase III Plus, FOXBase, NXPDB (NEXPERT's own proprietary database), SylkDB (Excel), and WKSDB (Lotus).

The second level of integration uses separate bridges to connect NEXPERT to the major relational databases on the market including Oracle, Sybase, INGRES, and Informix via Structured Query Language (SQL) queries. These database bridges allow NEXPERT to control the knowledge while a database management system manages the facts. There is a one-to-one mapping between tables, records, and fields in databases, and classes, objects, and properties in knowledge bases, respectively.

In the following example, NEXPERT passes the value of an object called *Pressure* to a file, executes a program which processes this data, and then retrieves the result from the program's output file.

```
IF Yes Process_Data  THEN  Execute_Program
            WRITE "input.nxp"@TYPE=NXP;@FILL=NEW;@ATOMS=Pressure.value;
            EXECUTE "calc.exe"@TYPE=EXE;@WAIT=TRUE;
            RETRIEVE "output.nxp"@TYPE=NXP;@ATOMS=Pressure.value;
```

Although NEXPERT has many powerful rule features, it lacks a facility for certainty factors. To provide for uncertainty in rule selection or user responses, the developer must create his own objects or variables to record and manipulate these certainty factors. Neuron Data has promised to correct this shortcoming in future versions of the product.

2.2 Objects

As the name implies, NEXPERT OBJECT is object-oriented in nature with facilities for classes, objects, methods, and inheritance. Groups of objects can be

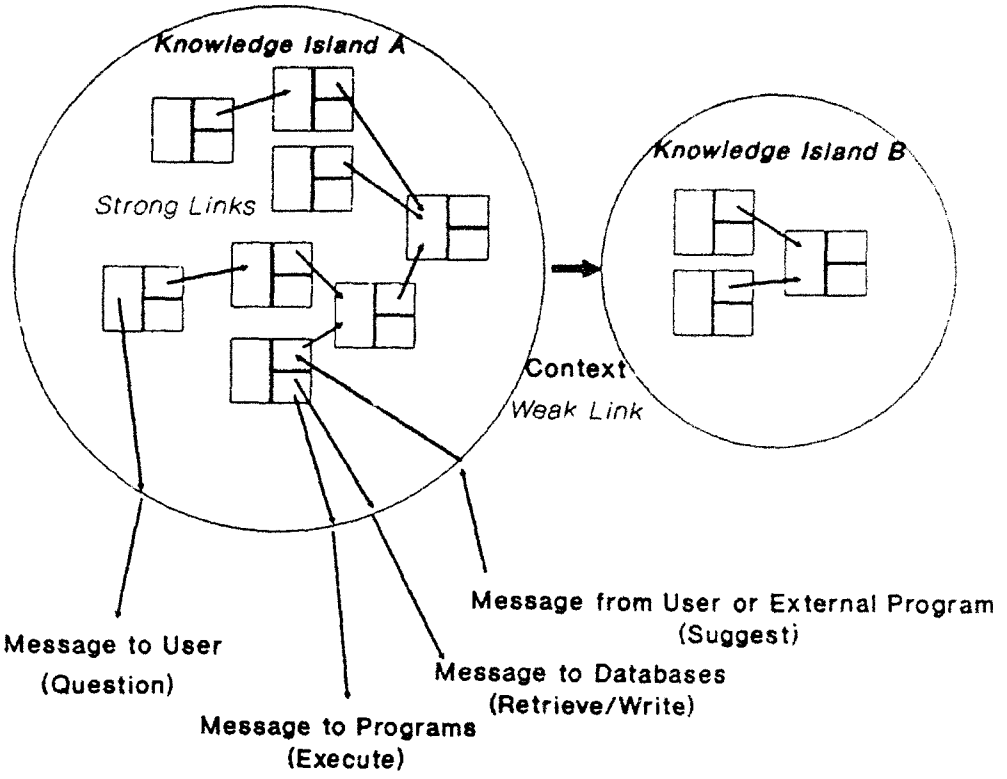


Figure 1: NEXPERT Knowledge Islands

networked into *knowledge islands*, as illustrated in Figure 1. All objects with *strong links* between hypotheses and conditions are grouped into the same knowledge island. *Weak links* connect knowledge islands. Weak links among rules and objects can be made manually at any time by the developer. In the example below, the inference engine has no reason to explore the *Condition_Red* hypothesis while the *Condition_Blue* hypothesis is under consideration, since there are no strong links among the conditions and hypotheses. *Condition_Red* would be considered only if a weak link was established to it, connecting the two separate knowledge islands.

```
IF Yes Boiler_damage                THEN Condition_Red
   IS  Boiler1 "Inoperative"

IF Yes Turbine_damage               THEN Condition_Blue
   IS  Turbine2 "Inoperative"
```

Groups of objects can also be categorized into classes. In the following example, inferences are made on the property *pressure* of a class *BOILERS*. If the condition is true, the *water_level* of all objects in the class is set to 5 and the property *setting* of an object called *Safety_valve* is set to "on."

```
IF IS |BOILERS|.pressure "critical" THEN  Shutdown
```

```
DO  |BOILERS|.water_level 5
LET Safety_valve.setting "on"
```

A *pattern* is closely related to a class. In the example below, any instance of the class *PUMP* that is "Off_Line" is a pattern. All objects belonging to that class whose properties match this pattern can be gathered into a *list*.

```
IF IS <PUMP>.status "Off_Line"      THEN  Notify_Operator
   >  <TURBINE>.rpm  3700
```

Through a combination of rules and objects, knowledge islands, and database links, NEXPERT provides a variety of methods for representing expert knowledge. In addition to its comprehensive knowledge representation strategies, however, NEXPERT supplies even more powerful reasoning and inferencies strategies.

2.3 Reasoning

In NEXPERT, objects may inherit properties from other objects or classes in a variety of ways. Inheritance can be changed at any time by using the built-in functions *InhValue Down*, *InhValueUp*, *InhMethod*, or *NoInherit*. Although most expert system shells provide only two methods of inferencing on rules, NEXPERT offers four: backward chaining, forward chaining, semantic gates, and context links. Backward and forward chaining are well-known among expert-system developers, and context links were explained above; gates are somewhat less well-known, however.

Semantic gates allow a rule net to pruned into a smaller search space. Through the use of semantic gates, the inference engine can selectively generate goal hypotheses during the evaluation of rules. For example, in Figure 2, condition 1 of the initial hypothesis is tested. Since this condition is true, a subsequent hypothesis which is connected to this condition will also be investigated at a later time. Condition 2, however, is false, so the hypothesis it falls under will not be investigated. By enabling or disabling these semantic gates, inferencing can proceed along a number of possible paths.

The complexity made possible through flexible inferencing and knowledge representation allows NEXPERT to predominate over competing products.

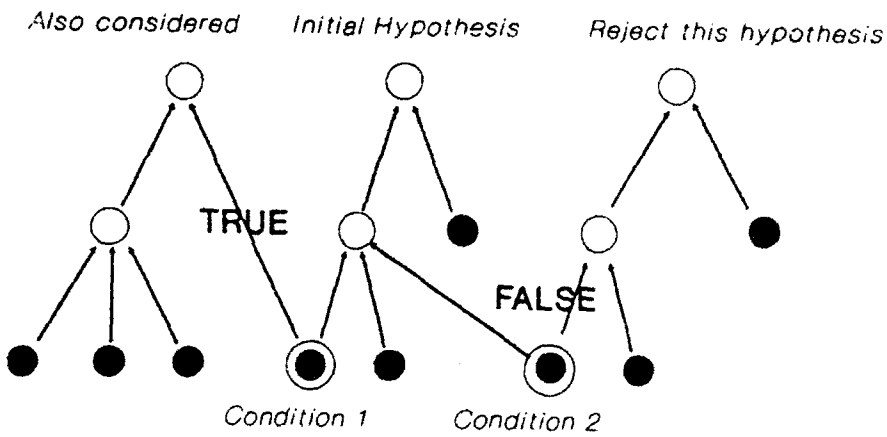


Figure 2: A Rule Net Illustrating Semantic Gates

3 Interface

NEXPERT's development and user interfaces present another competive advantage over other expert system shells. The emphasis on a professional, graphic interface allows the user to concentrate on the knowledge rather than the intricacies of the tool itself, providing more efficient and effective developement, maintenance, and use of the system.

3.1 Development Interface

NEXPERT's comprehensive graphic interface allows developers to edit rules and objects as well as build control structures, with an overview of the rule and object structures available at all times through a dynamic, graphic browsing mechanism. This interface offers a fast learning curve for the user, and the resulting knowledge bases are much easier to debug. These two qualities bring the domain expert closer to the system and reduce the need for a knowledge engineer.

In addition to its graphic knowledge editors, browsers, and windows, NEXPERT allows developers to playback a session through a journalling feature. Breakpoints can be set in the rule network, and various options and inferencing strategies can be explored through repetitve trials.

3.2 User Interface

In the end-user environment, NEXPERT provides support for how it came to a conclusion, why it is asking something during the consultation, and help for each question presented to the user. NEXPERT also provides a variety of prompts and menus for user input including a default menu and custom-designed screens through a tool called NORT (NEXPERT OBJECT Runtime).

By specifying *SHOW* as an action under the hypothesis of a rule, NEXPERT can display a variety of graphic formats including MacPaint, Dr Halo, Windows Paint, TIFF on IBM PCs, and PICT on MacIntosh. The same *SHOW* command can also display plain ASCII text.

4 Trial Run

The authors have developed several small expert systems using NEXPERT OBJECT and have encountered a few problems with its use.

Perhaps the greatest problem with NEXPERT is its lack of documentation. Although the product is extremely powerful and built with the end-user in mind, the necessary documentation to support its numerous features seems to have occurred only as an after-thought. Without adequate instructions on each of its features, much wasted time was spent on trial-and-error experiments. NEXPERT does provide some help over the phone, however. Lacking a detailed user manual with copious, practical examples, the best method of learning to use the product is probably through one of the training schools offerred by Bechtel, Inc. and a few other organizations.

Another nagging problem is the occasional bugs discovered with the product (although frequent updates of the software rapidly correct these). On several occasions the hardware locked up for no apparent reason (most of these problems dealt somehow with "garbage" piling up in memory).

Another less annoying problem is the poor access to classes and objects. By improving its object-oriented features, Neuron Data can present an even more competitively-differentiated product.

Neuron Data is committed to user support and ongoing development, however, and these problems should be corrected in upcoming versions of the software.

5 Conclusion

NEXPERT has a few rough edges and lacks certain desirable features, but generally is a powerful, comprehensive expert system shell. The combination of

networked objects, knowledge islands, and various inferencing strategies allow the tool to simulate the human mind to an extent largely unrivaled by competing expert system products. NEXPERT's powerful features and dominance assure it a leading position in the mid-range of knowledge-base tools.

NEXPERT's class and object structure, hardware support, and database links make it an excellent choice for some select applications. NEXPERT offers a rich choice of knowledge representation, inference strategies, and integration into other programs, but it is probably more appropriate for experienced expert system developers and users. If a potential user has access to an 80386-based machine with a megabyte-plus memory and is planning to develop large applications, NEXPERT is a good choice.

Appendix - Hardware and Software Requirements

NEXPERT OBJECT release 1.02 for the IBM PC microcomputer requires 640 KB of RAM (2 MB is recommended), a 1 MB hard-disk drive, and a 1.2 MB diskette drive. Other requirements include an EGA board and color monitor, a serial mouse, and an expanded-memory board with at least 2 MB of RAM recommended).

NEXPERT comes with a hardware key which must be installed in a serial port for proper operation of the program (the key is invisble to other peripherals using the port). NEXPERT checks for the hardware key occasionally; if it is missing, the program will exit to DOS.

The PC version requires Microsoft Windows to support its windowing features.

Acknowledgements

A review of NEXPERT OBJECT was made possible by a grant from the Army Institute of Research in Management Information, Communications, and Computer Science (AIRMICS), Altanta, GA. Grant #: DAKF-11-88-C-0021.

The Authors

Milam W. Aiken received the B.S. degree in Engineering and the Master's of Business Administration degree from the University of Oklahoma and the B.A. degree in Computer Science and the B.S. degree in Business from the State University of New York. He is currently a Ph.D. student in Business Administration with a major in MIS at the University of Arizona, and his research interests include expert systems, office automation, and group decision support systems.

Olivia R. Liu Sheng received the B.S degree from the National Chiao Tung University in Taiwan, R.O.C. and the Master's and Ph.D. degrees in Business

Adminsitration with a major in Computers and Information Systems from the William E. Simon Graduate School of Business Adminsitration, University of Rochester, Rochester, N.Y. She is an assistant professor of MIS at the University of Arizona. Her principal research interests are analysis and design of distributed information systems.

Appendix V

Software Review: EXSYS Professional

Milam W. Aiken

Department of Management Information Systems
College of Business and Public Administration
University of Arizona
Tucson, AZ 85721
602-621-2748

Abstract

EXSYS Professional is an expert system system shell developed by EXSYS, Inc., P. O. Box 11247, Albuquerque, NM, USA 87192-0247 Phone: (505) 256-8356 FAX: (505) 256-8359. EXSYS Professional version 3.2 running under MS-DOS (reviewed in this article) costs $795, ($516.75 with the 35% educational discount). EXSYS Professional is also available for VAX/VMS- and Unix-based systems and is fully compatible across operating environments.

1 Introduction

Since 1985, when the original version of EXSYS was introduced, thousands of expert systems have been developed in academic and commercial environments using this product. This expert system shell has found wide-spread acceptance due to its power, portability, speed [1], and ease-of-use [6]. EXSYS Professional has all of the features of standard EXSYS while adding numerous additional capabilities which are detailed in this article.

2 Knowledge Representation and Inferencing

Like many expert system shells, EXSYS represents its knowledge in the form of IF-THEN production rules [2]. With a color monitor, the conditions of a rule which are true, false, or unknown can be shown in different colors *(Figure 1)*.

EXSYS allows forward chaining, backward chaining, or both. A particularly useful feature added in the Professional version is the *UNDO* command which allows the user to go back and correct a mistake. It is very frustrating for the user to answer dozens of questions during an expert system consultation, make a mistake, and have to start over from scratch. By simply pressing *CTRL-U*, the user can back up to the previous question and enter the correct information. Backtracking with *UNDO* is allowed up to 10 times.

[1] In a recent review of nine microcomputer expert system shells, EXSYS was consistently ranked among the fastest in terms of execution, compiling, loading, and saving [3].

[2] Knowledge representation through frames is available from an add-on product called FRAME developed by California Intelligence.

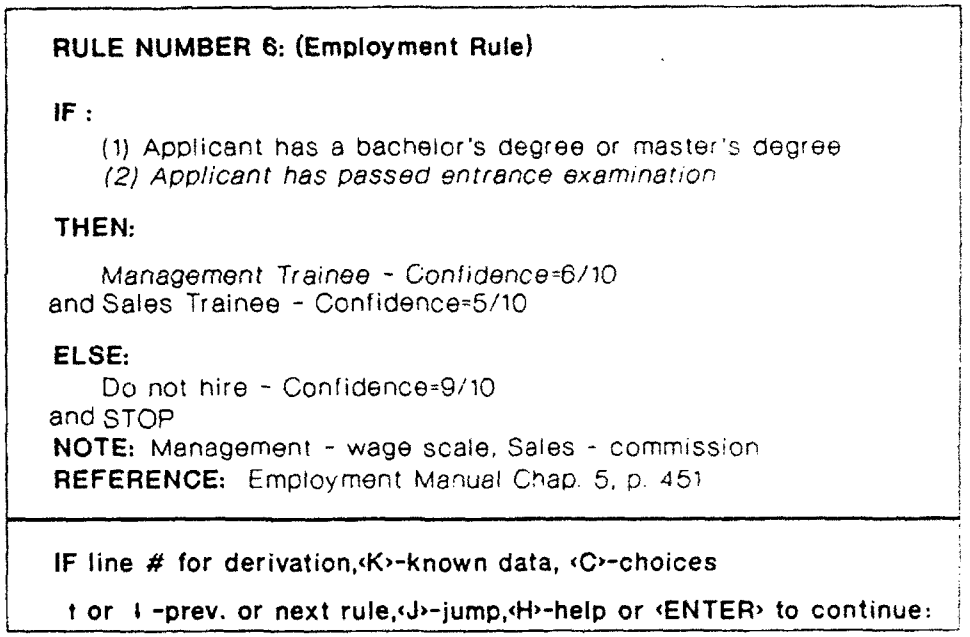


Figure 1: A Sample Production Rule

In addition to the average, independent, and dependent methods for combining certainty factors provided by the original version of EXSYS, Professional adds increment/decrement and user-defined formulas. Certainty factor values may range from 0 to 10, -1 to 1, or -100 to 100. With this large variety of inferencing and certainty factor strategies, very robust knowledge bases can be built.

3 Interface

An expert system's interface is perhaps the most critical aspect for user acceptance. If the system is not user friendly, does not respond quickly, or does not provide information in a useful format, the likelihood that the system will be used is slim. EXSYS Professional provides many important features for the user interface (development and runtime), the file interface, and the program interface.

3.1 User Interface

EXSYS Professional is divided into two principal programs — EDITXSP for knowledge base development and EXSYSP for running the finished expert system. The development interface in EDITXSP allows the user to inference through the rules (like EXSYSP), but includes many questions and features which may unnecessarily distract the user who wishes to only run the system.

To begin a knowledge base with EDITXSP, the developer simply answers a series of questions regarding entry and exit screens, application of certainty factors, execution of external programs, etc. (default options allow the inexperienced user to begin a knowledge base without having to understand all of the parameters involved) before the main development screen is shown *(Figure 2)*. The user then edits the knowledge base rules and conditions using Enlish text, menu selections, or algebraic expressions. As each new rule is entered, EXSYS automatically compares the new rule against existing rules and displays possible conflicts. Finally, the developer may provide an edit password which gives end users complete

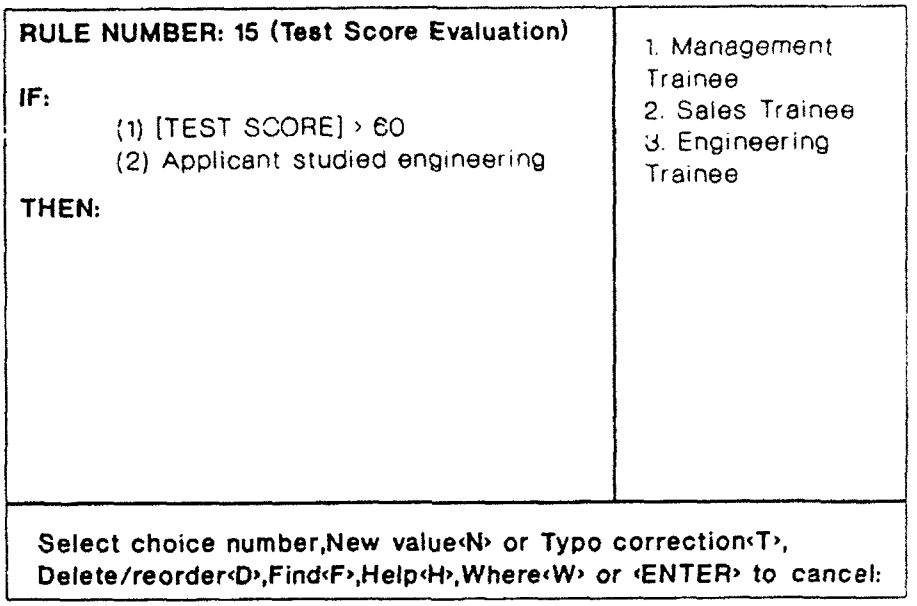


Figure 2: Developing the Knowledge Base

access to the knowledge base or a run password which allows end users to run the system but not to modify it. End users may also be restricted from viewing the rules, if necessary.

For knowledge base developers wishing to use a wordprocessor, a rule compiler is available to translate the rules into EXSYS' internal compiled format. This method also allows developers to transfer knowledge bases from one expert system shell format into the EXSYS format.

Running the expert system using EXSYSP is extremely easy. The user may select one or more choices from a menu to fulfill a condition or may enter a value when the system is asking about a variable. The default runtime screen is shown in *Figure 3*. This screen may be customized, however, with a screen definition language to provide color, borders, multiple screens, etc. Even more complex screens for input and output can be provided by calling an external program (see *program interface* below).

At the end of the session, a report summary is shown which indicates the expert system's recommendations ranked by certainty factor *(Figure 4)*.

3.2 File Interface

EXSYS' report generator allows session results and conclusions to be passed to other programs via an ASCII file. In addition, a sequential set of input data can be analyzed record-by-record.

Another type of file interface provided by EXSYS Professional allows the system to directly read and write to Lotus 1-2-3 and Ashton-Tate's dBase file structures. Using the command language (discussed below), an expert system can step through each record of a dBase file, analyze the data, and write the results to the dBase file or a separate file.

3.3 Program Interface

EXSYS Professional can be customized by calling external programs. Special graphics, numerical analysis, simulation, and interface programs can be run to add additional power to the expert system shell. Any external program can be called including .EXE, .COM, and .BAT files. In addition, interpreted programs

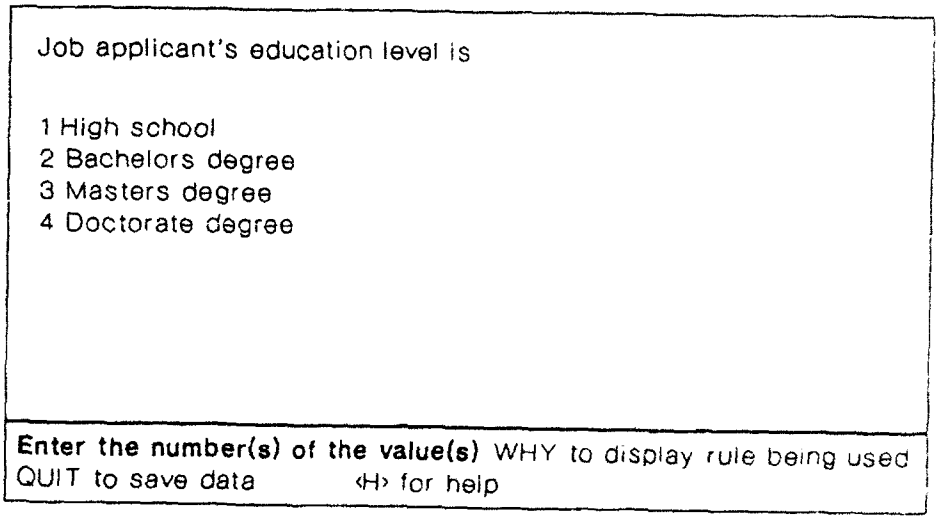


Figure 3: Running the Expert System

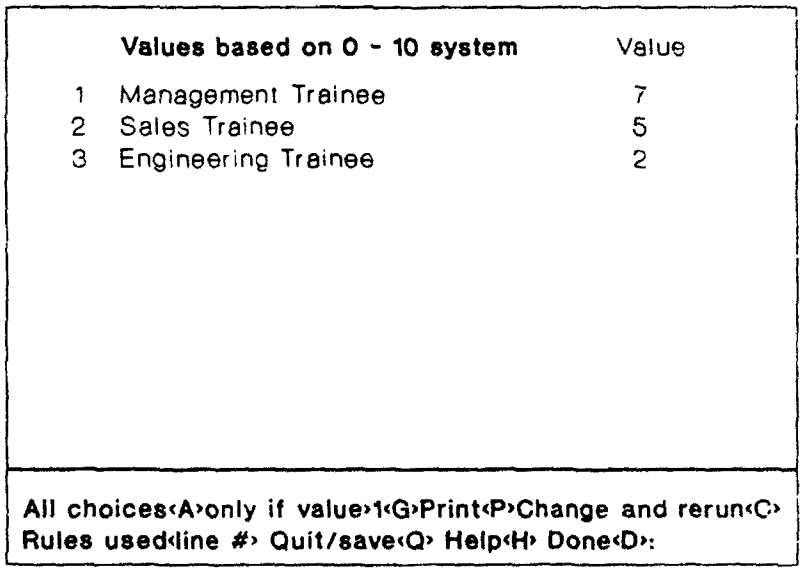


Figure 4: Report Summary

```
/* A Sample EXSYS Professional Command Language File
rules section1          /* Apply named subset of rules
if [Y] = "END"          /* conditional branch on a variable
   goto end
endif
/* Look through dBase III file called group.dbf
set [I] 1
while ([I] < toprec("group.dbf"))
   db_gn(group.dbf,[I],project,[Z],individual,[W])
   if [X] = [Z]  /* matched project
      report group.out    /* write result in file group.out
   endif
   set [I] ([I]+1)        /* increment record counter
wend
:end
report summary          /*  Output in file called summary
run final.exe summary "End Report" /C  /* Call external program
exit                    /*  Exit to DOS
```

Figure 5: Sample Command Language File

may be called by first calling the interpreter and then the program (e.g., BASIC CALC.BAS). Parameters and data can be passed to and from the called program directly (through variables or the command line) or indirectly (through an intermediate ASCII file).

4 Command Language

A major additional feature provided by the Professional version of EXSYS is the knowledge base command language (use is optional). The command language is similar to a batch language and provides detailed control of knowledge base execution, input of data, looping, and display of results *(Figure 5)*. This is particularly important when used in conjunction with invisible embedding of the expert system with other applications for real-time process control. Named rule subsets in the command language also provide modular control over the knowledge base.

5 Learning to Use EXSYS Professional

EXSYS Professional is so easy to learn and use that even novices with no prior experience in knowledge base development are usually able to get a significant expert system up and running in a couple of days. Extensive use of menus and context-sensitive help directs the user through the development and use of the tool. Additional help is provided through a comprehensive tutorial on five diskettes. The user manual is extremely well-written with detailed cross-referencing and indexing. Also, several books have been written which illustrate the use of EXSYS [1, 2, 4, 5]. And finally, training seminars are offered by many academic institutions and a few software development firms.

6 Conclusion

Few expert system shells available for microcomputers combine as much power for as little cost as does EXSYS Professional. EXSYS Professional offers advantages in terms of efficiency (speed and memory utilization), ease of learning and use, and cost.

By providing powerful, relatively inexpensive, and easy-to-use expert system shells directly to the domain expert, the knowledge acquisition bottleneck can be broken. In the academic environment, students can be expected to learn to use the tool themselves, allowing the instructor to concentrate on teaching the theory of knowledge engineering.

Both novices and experienced knowledge engineers are encouraged to explore the capabilities of this powerful product. For only $15, a demonstration version of the original EXSYS (limited to saving only 25 rules to disk) is available. The purchase price includes a series of tutorial lessons, several sample expert systems illustrating various aspects of the tool, and a manual. A useful prototype expert system can be built using this restricted version of the tool and can serve to convince the user or management of the effectiveness of EXSYS and EXSYS Professional. At such a low price, there is little risk in investigating the product.

References

[1] Girard, J., Carico, M., and Jones, J., *Building Knowledge Systems Using Rule-Based Shells*, McGraw-Hill.

[2] Murray, J., *Expert Systems in Data Processing: A Professional Guide*, McGraw-Hill, 1988.

[3] Press, L., "Eight-Product Wrap-Up: PC Shells," *AI Expert*, September 1988, pp. 61-65.

[4] Ruth, S. and Sprague, K., *Developing Expert Systems: Using EXSYS*, Mitchell Publishing.

[5] Turban, E., *Decision Support and Expert Systems: Managerial Perspectives*, MacMillan Publishing.

[6] Vedder, R., "PC-Based Expert System Shells: Some Desirable and Less Desirable Characteristics," *Expert Systems*, February 1989, Vol. 6, No. 1, pp. 28-42.

Appendix - Hardware and Software Requirements

EXSYS Professional for the IBM PC, XT, AT or compatible requires a minimum of 640K RAM, a hard disk or high density floppy disk drive, and DOS 2.0 or higher. The program will work with either a color or monochrome monitor, but color allows the user to see what conditions of a rule are true, false, or unknown when inferencing.

EXSYS Professional uses all main memory that is available. If the knowledge base does not fit, rules are swapped to and from the hard disk as necessary. Approximately 64K is necessary for 500 rules with an average of six or seven conditions in each rule.

Acknowledgements

A review of EXSYS Professional was made possible by a grant from the Army Institute of Research in Management Information, Communications, and Computer Science (AIRMICS), Atlanta, GA. Grant #: DAKF-11-88-C-0021.